

User Review Analysis for Requirement Elicitation

JUAN WANG

王娟

Thesis submitted in partial fulfilment of the requirements of the award of

Doctor of Philosophy

in

School of Engineering, Computing and Mathematics

Faculty of Technology, Design and Environment

Oxford Brookes University, Oxford, UK.

June 25, 2020

Abstract

Online reviews are an important channel for requirement elicitation. However, requirement engineers face challenges when analysing online user reviews, such as data volumes, technical supports, existing techniques, and legal barriers.

This thesis proposes a framework solving user review analysis problems for the purpose of requirement elicitation that sets up a channel from downloading user reviews to structured analysis data.

This framework is believed to be able to solve the problems because (a) the structure of this framework is composed of several loosely integrated components, which not only realize the flow of data from downloading raw user reviews to the structured analysis results, but also provide adaptability and flexibility for wider future applications; (b) the reasonable use of linguistic rules makes it possible to adjust and control the internal details of the system in this data flow; (c) natural language processing (NLP) technologies, such as chunking, regular expressions, and especially Stanford dependency trees, provide substantial technical support for this framework.

Three mobile app user review datasets were used to evaluate the functionalities. 6081 user reviews from the first dataset is used for the development of linguistic rules. The first two datasets are used to enrich the popular opinions and the keywords list. The third dataset acts as a control group. The performance results of the prototype demonstrate that this framework is practical and usable.

The main contributions of this work are: (1) this thesis proposed a framework to solve the user review analysis problem for requirement elicitation; (2) the prototype of this framework proves its feasibility; (3) the experiments prove the effectiveness and efficiency of this framework.

Keywords: user review analysis, requirement elicitation, mobile app user reviews, Google Play, user request elicitor, topic-opinion extractor, Stanford Dependency Parser, ontology populator, general mobile app user review ontology.

Declaration

This thesis is submitted to the Oxford Brookes University in accordance with the requirements of the award of Doctor of Philosophy in the Faculty of Technology, Design and Environment. It has not been submitted for any other degree or diploma of any examining body. Some parts of the work presented in this thesis have previously appeared in the published papers listed in the publications section. Except where specifically acknowledged, it is all the work of the Author.

Juan Wang, June 25, 2020.

王娟, 2020年6月25日

Acknowledgements

When time comes to the moment of acknowledgements, I become voiceless and shed tears. Only people who know what happened would understand my reasons. For people who don't know, I would like you to remember a happy smile in the future.

My utmost gratitude is to my parents. You gave me this life and the bravery to live it regardless of what is ahead. I am so sorry for the troubles and difficulties that I had brought to you. This PhD, which I tried my very best to get, is my gift to you.

My heartfelt gratitude is also to my older brother, sister-in-law, and my nephew. Thanks for your support, your understanding, and your encouragement. Special thanks to my older brother, for your blood-tied sincere care and good wishes.

My special appreciation has to go to my Director of Studies, Dr. Tjeerd Olde Scheper, who saved my PhD multiple times from the very dark moments. His support not only made this PhD become possible, but also avoided various dangers of abnormal failures. The words of thanks are never sufficient for his support and help. His justice will illuminate the road ahead of me.

My special appreciation is also with my enlightening teacher in computing, Dr. David Sutton. Dr. Sutton started to teach me in an Object-Oriented Programming course six years ago. He is the first lecturer who made me mindful of basic programming. From this starting point, he kept his help whenever I asked for all these years. For countless times, he saved me out of deadlocks. Bit by bit, I learn from him the methods, the trains of thoughts for breaking deadlocks. He not only showed me how to look for knowledge, how to do troubleshooting and resolve problems, but also presented me with his charitable way of helping students. Looking back at his six years of help, I am in tears.

My sincere appreciation must also be addressed to Prof. Hong Zhu. As his teaching assistant for several modules, I learned knowledge from his courses and the values of life from him. He can use a couple of sentences to point out a direction for me, either for research or for life.

My genuine appreciation is also with my other supervisors, Prof. Rachel Harrison, Dr. Arantza Aldea and Dr. Clare Martin. They have been a great help for all these years as well.

My sincere salutes are to my adviser, Prof. Maricela Bravo, for her voluntary help to this PhD with her expertise.

Many professors, lecturers and professionals helped this PhD with their expertise, Nicky Holland, Gerald Roy, Katie Hambrook, Daniel Croft, Carlos Fresneda-Portillo, Peter Ball, Jill Organ, Chrisina Jayne, Nigel Crook, Mike Joy, and Sam Varney, who have been willing and able to give me support in their domains. All these names are here for a significant reason, especially Nicky Holland.

Many friends deserve my many thanks because they cast light into my difficult journey along all these years. You probably changed my direction by a simple suggestion. You perhaps saved me out from a nightmare just because of a sentence. I will shout to the sky with all my strength to let you know my appreciation and happiness for ever meeting you in my life.

Table of Contents

<i>Abstract</i>	<i>i</i>
<i>Declaration</i>	<i>ii</i>
<i>Acknowledgements</i>	<i>iii</i>
<i>Table of Contents</i>	<i>v</i>
<i>List of Figures</i>	<i>x</i>
<i>List of Tables</i>	<i>xii</i>
<i>List of Code Snippets</i>	<i>xv</i>
<i>List of Algorithms</i>	<i>xvi</i>
<i>Abbreviations</i>	<i>xvii</i>
<i>List of Code in the CD</i>	<i>xix</i>
1 Introduction	1
1.1 Background	1
1.2 Thesis statement and objectives	4
1.3 The original contributions	4
1.4 Legal considerations for downloading reviews and their publishing	5
1.4.1 Copyrights and the Copyright, Designs and Patents Act 1988	5
1.4.2 Personal data, GDPR and the Data Protection Act 2018	6
1.4.3 Conclusions from the legal considerations	8
1.5 Top level structure	8
1.6 Research methodology	11
1.7 Thesis structure	12
2 Literature review	13

2.1	Opinion mining	13
2.1.1	A traditional opinion mining model.....	16
2.1.2	Co-extraction.....	24
2.1.3	Sentiment orientation analysis.....	34
2.1.4	Fake reviews.....	37
2.2	NLP in requirement elicitation	39
2.3	Concept mapping onto ontologies	46
3	Methods	49
3.1	A brief description of the architecture of the system	50
3.2	Web sourcing user reviews	52
3.2.1	Web sourcing process.....	52
3.2.2	Mobile app page metadata sourcing.....	56
3.2.3	Sourcing the full review pages.....	62
3.2.4	Controlling the scrollbar.....	64
3.2.5	Dealing with emoji.....	67
3.2.6	Methods dealing with issues appearing during the code running.....	68
3.3	SO-CAL and its slight adjustments in this framework	72
3.3.1	Creating Windows Batch files.....	72
3.3.2	A few fixes of SO-CAL in the new environment.....	73
3.3.3	Adjustments to enable writing sentences and sentiment scores to the database.....	75
3.3.4	Steps to run SO-CAL in this framework.....	78
3.4	Automatic user requests elicitation	82
3.4.1	Proposing generalized grammar rules and their trade-offs.....	82
3.4.2	Tackling the data sparsity problem.....	83
3.4.3	An example process of mining user requests linguistic rules.....	84
3.4.4	Single rule structures and flexibilities.....	88

3.4.5	The current coding structure for user request linguistic rules.....	91
3.4.6	Two example rules inspired by English grammar	93
3.4.7	The suggested way to mine the rules and enrich them in the future.....	94
3.4.8	Attention that might save time for future researchers.....	96
3.5	Topic-Opinion extraction.....	98
3.5.1	A simple example for how this component works for a dependency tree	99
3.5.2	Topic-Opinion pair extraction via linguistic rules.....	103
3.5.3	Dealing with negation words.....	107
3.5.4	The algorithm in version one of the traversal function.....	109
3.5.5	The algorithm in version two of the traversal function	112
3.5.6	Topic-Opinion pair filtering via keywords	121
3.5.7	Data flow connector between the Request Elicitor and the Topic-Opinion Extractor.....	125
3.5.8	The debugging code to help future adjustments	127
3.5.9	Explicit opinions plus a little bit of implication for the “app” topic.....	129
3.6	General Mobile App User Review Ontology (GMAURO)	132
3.6.1	Ontology design	132
3.6.2	An algorithm for complex ontology population	138
3.6.3	Methods to break down large ontology populations.....	140
3.6.4	Ontology population code structure	142
3.6.5	How to read existing individuals from the ontology if they exist.....	143
3.6.6	Merging negation words with opinion words.....	144
3.6.7	Overcoming the data problems that crash the ontology	144
4	<i>Results and evaluation</i>.....	147
4.1	Evaluation methods.....	147
4.2	A checklist for the experimentation.....	150
4.3	Performance results	154
4.4	Interpretation of the results.....	157

4.5	A quick evaluation for the negation words handling	170
4.6	Several SPARQL queries about some applications of the GMAURO ontology 174	
5	<i>Discussion</i>	191
5.1	Summary of this thesis	191
5.2	Adaptability of this prototype	192
5.2.1	Measures	192
5.2.2	Contrary linguistic rules.....	193
5.3	Advantages and disadvantages of each component and the framework.....	195
5.4	Flexibility of this framework	197
5.4.1	Flexibility in component level	197
5.4.2	Flexibility in logics and algorithms level.....	198
5.5	A minor bug in the topic-opinion extractor traversal function version two.	199
5.6	Future research directions.....	203
6	<i>Conclusion</i>	205
7	<i>References</i>	207
8	<i>Appendix A Software installation</i>	221
8.1	Firefox 47 installation [128].....	222
8.2	The geckodriver installation.....	223
8.3	MySQL, and supporting Visual Studio, Python 3 (64-bit) installation	223
8.4	Python 3 (32-bit) and libraries such as NLTK installation	224
8.5	Selenium installation	225
8.6	graphviz installation.....	226
8.7	Stanford Parser installation	226
9	<i>Appendix B Steps to run this framework</i>	228

<i>10</i>	<i>Appendix C POS tag list and user request rule convention</i>	<i>229</i>
<i>11</i>	<i>Appendix D Topic-Opinion pair extraction linguistic rules</i>	<i>233</i>
<i>12</i>	<i>Appendix E Popular Opinions extracted from dataset one and two.</i>	<i>239</i>
<i>13</i>	<i>Appendix F OOPS!'s evaluation result for GMAURO</i>	<i>248</i>
<i>14</i>	<i>Appendix G Answers to the SPARQL queries about the applications of the GMAURO ontology.....</i>	<i>256</i>

List of Figures

Figure 1 The framework architecture: Batch Sequential Data Flow.	9
Figure 2 An example in the top level logic of this framework	10
Figure 3 Example of an opinion with an opinion holder	17
Figure 4 Example of an opinion without an opinion holder	17
Figure 5 The framework architecture with further more detail	50
Figure 6 Metadata being mined on a typical mobile app page	57
Figure 7 Icon for the tool locating webpage elements	57
Figure 8 An example of the input file containing a list of apps.....	60
Figure 9 Review data being mined on a typical review	63
Figure 10 User requests linguistic rules manual mining process.....	83
Figure 11 Topic-Opinion extraction process.	98
Figure 12 A dependency tree	100
Figure 13 A dependency tree with arcs that match linguistic rules highlighted	101
Figure 14 A dependency tree with the arcs that pass filtering highlighted.....	102
Figure 15 Negation word is a child of the opinion	108
Figure 16 Negation word is a sibling of the opinion.....	108
Figure 17 Negation word is the parent of the opinion	109
Figure 18 Example sentence containing a root orphan opinion.....	114
Figure 19 Example sentence containing a root opinion in a pair.....	114
Figure 20 An example of orphan opinion	119
Figure 21 Examples of interference between orphan opinions and pairing rules.....	120
Figure 22 Observed decline in reacted rate for two-word sentences	120

Figure 23 Example result of Topic-Opinion pair extraction.....	125
Figure 24 GMAURO class hierarchy.....	133
Figure 25 The inverse relationships among the six object properties.....	134
Figure 26 A reasoner highlights the inferred type of a review	135
Figure 27 Object properties and data properties in GMAURO.	136
Figure 28 Easy navigations in the ontology.....	137
Figure 29 User review compositions.....	148
Figure 30 Reasons for FPs and FNs and the percentages in three datasets.....	163
Figure 31 The pitfalls that OOPS! found for GMAURO.....	175
Figure 32 Examples of contrary linguistic rules' usage.....	194
Figure 33 Stanford NLP Group explains the JDK version issue	227
Figure 34 A screenshot of an example result of SPARQL query Q1	257
Figure 35 A screenshot of an example result of SPARQL query Q2	258
Figure 36 A screenshot of an example result of SPARQL query Q3	259
Figure 37 A screenshot of an example result of SPARQL query Q4	261
Figure 38 A screenshot of an example result of SPARQL query Q6	262
Figure 39 A screenshot of an example result of SPARQL query Q7	264
Figure 40 A screenshot of an example result of SPARQL query Q8	265
Figure 41 A screenshot of an example result of SPARQL query Q9	266
Figure 42 A screenshot of an example result of SPARQL query Q10	267
Figure 43 A screenshot of an example result of SPARQL query Q11	269
Figure 44 A screenshot of an example result of SPARQL query Q12	269

List of Tables

Table 1 Different focuses of surveys in sentiment analysis	14
Table 2 Research of mining or refining opinions	17
Table 3 Research contributed to mining opinion targets (topics)	19
Table 4 Research contributed to identifying opinion holders	20
Table 5 Research of bridging topics and opinions together.....	22
Table 6 Papers surveyed for topics and opinions co-extraction.....	24
Table 7 Some papers on sentiment orientation classification	35
Table 8 A few typical research on fake reviews	37
Table 9 Papers using natural language processing in requirement elicitation	40
Table 10 an example inclusion criteria formula.....	55
Table 11 Included driving theory mobile apps from the third dataset	55
Table 12 XPath values of typical metadata on 27th December 2019	58
Table 13 XPath values of typical review data on 27th December 2019	63
Table 14 JavaScript controlling scrollbar to an element.....	64
Table 15 Methods to ignore or to replace emoji with "?"	68
Table 16 SQL queries dealing with unresponsive script issue if the exception block is triggered	70
Table 17 Initial keyword combinations and their occurrences in dataset one	85
Table 18 Request analysis for “could you + <VB>”	85
Table 19 A sample rule for “could you + <VB>”	87
Table 20 One clue leads to two linguistic rules covering 66 occurrences in dataset one	87
Table 21 Rule 100: wish + past tense simple verb or a modal verb	93

Table 22 Rule 87: Maybe + a verb or an adjective	94
Table 23 An example of 'UserInterface' in the two-dimensional keywords array	123
Table 24 Frequencies of "app" and related topics across three datasets	130
Table 25 Equivalent classes of three types of reviews.....	135
Table 26 Granting privileges to the role in the database connection	143
Table 27 Performance of the first samples of three datasets.....	155
Table 28 Reasons for FPs and FNs in the three datasets.....	158
Table 29 The explanations for the reasons of FNs and FPs.....	160
Table 30 Association between the reasons and the treatments for Dataset one.....	164
Table 31 Association between the reasons and the treatments for Dataset two.....	165
Table 32 Association between the reasons and the treatments for Dataset three.....	166
Table 33 Maximum performance impact prediction after the fixes for three datasets	168
Table 34 Performance of negation handling in three datasets	170
Table 35 Definitions of various results in negation evaluation	171
Table 36 An Excel formula to check whether a review sentence contains a negation word	173
Table 37 Time taken by Snap SPARQL when answering the questions	177
Table 38 An example of Snap SPARQL Query for Question 1.	180
Table 39 The answer to Question 1 from dataset one.....	181
Table 40 The answer to Question 1 from dataset two.....	181
Table 41 The answer to Question 1 from dataset three.....	181
Table 42 An example of Snap SPARQL Query for Question 2.	182
Table 43 The answer to Question 2 from dataset one.....	182
Table 44 The answer to Question 2 from dataset two.....	184

Table 45 The answer to Question 2 from dataset three.....	188
Table 46 Calculations of the negative impact of this bug on precisions.....	200
Table 47 Software versions that work in this implementation.....	221
Table 48 Steps to run this system.....	228

List of Code Snippets

Code Snippet 1 Code pointing to FireFox 47	53
Code Snippet 2 Code scrolling the web page downwards	53
Code Snippet 3 Code producing a csv spreadsheet with the data	54
Code Snippet 4 Getting elements by group or individually	57
Code Snippet 5 Getting “update date” by XPath and converting to “date” type for MySQL	61
Code Snippet 6 Outputting metadata to a text file	61
Code Snippet 7 Outputting metadata to the database.....	62
Code Snippet 8 Method getting “id” of the existing review for the associated developer reply	65
Code Snippet 9 Fix for preprocess.py	73
Code Snippet 10 SO_Run.py line 237- 244 change.....	75
Code Snippet 11 Code writing sentences and sentiment scores to the database.....	76
Code Snippet 12 Creating the required review files and folders for SO-CAL	79
Code Snippet 13 An example chunking and chinking grammar	88
Code Snippet 14 An example regular expression test.....	89
Code Snippet 15 An example of flexibility in current user request rule structure.....	90
Code Snippet 16 SQL query fulfils the data flow connector between request elicitor and topic-opinion extractor	127
Code Snippet 17 The method to execute the population algorithm item by item.....	140
Code Snippet 18 The method to break down records of one item in the population.....	141
Code Snippet 19 The code relates to this bug	199

List of Algorithms

Algorithm 1 Pseudocode of the loops controlling the scrollbar on the full review page.....	66
Algorithm 2 Automatic user request elicitation	91
Algorithm 3 The traversal function version one	110
Algorithm 4 Simplified pseudocode of the traversal function version two	115
Algorithm 5 Algorithm for complex ontology population.....	140

Abbreviations

API	application program interface
Brookes RADAR	Research Archive and Digital Asset Repository (the institutional repository of Oxford Brookes University)
CSS	Cascading Style Sheet
CQ	Competence Questions for ontology evaluation
EEA	European Economic Area
FN	False Negative
FP	False Positive
GATE	General Architecture for Text Engineering tool
GDPR	The EU General Data Protection Regulation
GMAURO	General Mobile App User Review Ontology
IDLE	Integrated DeveLopment Environment for Python
IR	Information retrieval
IRI	Internationalized Resource Identifier
NLP	Natural Language Processing
NLTK	The Natural Language Toolkit
POS	Part-Of-Speech
OOPS!	Ontology Pitfall Scanner
RE	Requirement Engineering

SBVR	Semantic Business Vocabulary and Rules
SE	Software Engineering
SO-CAL	Semantic Orientation CALculator for Sentiment Analysis
TN	True Negative
TP	True Positive
URL	Uniform Resource Locator
WAMP	Wamp Server (Windows, Apache, MySQL, and PHP)

List of Code in the CD

ID	File name	Content
C.1	C1-DatabaseCreation.txt	Database creation statements
C.2	C2-GooglePlayAppsSelection.py	Web sourcing code to get metadata for all apps returned on a search result page.
C.3	C3-DownloadAppFirstPage.py	Web sourcing code for mobile app page metadata
C.4	C4-DownloadFullReviewsFromListApps.py	Web sourcing code for mobile app all reviews on full review page
C.5	C5-convertDBreviewsToFilesOnce.py	Converting reviews from database to text files for SO-CAL.
C.6	C6-posAsentence.py	Code to parse a sentence into Part-Of-Speech tags
C.7	C7-chunkwithchinking®ualarExpressions-Debugging.py	Debugging code to help user request linguistic rules' creation
C.8	C8-chunkwithchinking®ualarExpressions.py	User request elicitation code
C.9	C9-stanfordDependencyTreeV1.py	Topic-Opinion pair extraction via linguistic rules Version one
C.10	C10-stanfordDependencyTree+orphansV2.py	Topic-Opinion pair extraction via linguistic rules Version two

C.11	C11-keywordComparison+wordnetV1.py	Topic-Opinion pair filtering via keywords Version one
C.12	C12-keywordComparison+wordnetV2.py	Topic-Opinion pair filtering via keywords Version two
C.13	C13- stanfordDependencyTreeDebuggingV1.py	Debugging code to help topic-opinion pair extraction and filtering Version one
C.14	C14- stanfordDependencyTreeDebuggingV2.py	Debugging code to help topic-opinion pair extraction and filtering Version two
C.15	C15-The ontology populator source code (a Java project)	The ontology populator source code (a Java project)
C.16	C16-sampleEvaluationDataPerReview.py	Python code to produce evaluation data spreadsheet.
C.17	C17-posOpinions.py	Python code to help the adaptation of popular opinions.
	UserRequestLinguisticRules.xlsx	User Request Linguistic Rules management spreadsheet
	generalMobileAppUserReviewOntologyV ersion14.owl	The ontology definition in this prototype

1 Introduction

This chapter introduces the key points that instruct this thesis in research direction and structure. Section 1.1 introduces the background that helps to inform the research question. Section 1.2 reports the aim of this PhD and the objectives. Section 1.3 briefly declares the solution and the novel contributions. Section 1.4 justifies that the usages of the web sourcing code in this thesis is legal through a series of reasoning to answer the relevant legality questions. This section also draws conclusions on whether to retain the user reviews data downloaded in this thesis and whether it can be legally made public. Section 1.5 describes the top level logic of this framework with an example in a picture. Section 1.6 briefly reports how this framework is informed in terms of research methodology. Section 1.7 iterates the thesis structure in advance.

1.1 Background

Online user reviews contain a wealth of user requirements and are natural expressions of the opinions and needs from many users around the world. They effectively reflect the users' requirements with the amount of data and the characteristic of collecting data sources across the world. Therefore, user reviews are valuable sources of requirements in comprehensiveness and referential values. Especially, a wide range of sources and large data volumes make online reviews an inevitable research topic for requirement engineers.

Although some studies have shown that some of the reviews are fake [1]–[5], so far there has been no research to claim that the fake reviews have reached a threshold that can make the overall online reviews lose their research value.

Traditional requirement elicitation channels are mainly the interactions between the requirement engineers and the specific users in the forms of documentations, interviews and questionnaires. These methods usually consume high labour hours, long cycles of time, and are often subjectively affected by requirement engineers. The effectiveness of the traditional requirement elicitation methods therefore encounters a bottleneck in front of the present online user reviews.

Online user review analysis can overcome the above limitations of traditional requirement elicitation. Due to its low cost, objectivity, and comprehensive nature, it could become an important supplement to traditional requirement elicitation methods. However, requirement engineers face at least four challenges when analysing online user reviews.

Firstly, the amount of data for online reviews is not accomplishable for human brains.

Secondly, the online reviews platforms provide limited technical support for requirement engineers to perform reviews analysis. For example, Google Play offers a developer API for app owners [6]. The methods that are offered by the API are only limited to listing the full texts of the user reviews and replying to users [6]. It will be difficult for the developers if they intend to have an insight into these user reviews for more information via the provided APIs.

Thirdly, the existing technologies have not been fully utilized to analyse the user reviews. Many researchers have been working continuously to develop new technologies and improve existing technologies, but some of them still use previously existing datasets. This makes many good technologies unavailable for practical use.

Fourthly, legal barriers make it impossible for business researchers to legally obtain user reviews of products or software that are not their own. This is discussed in the later subsection 1.4 for related laws currently effective in the UK.

However, there is a considerable number of commercial companies, such as App Annie [7], Apptopia [8], etc., who do services for mobile app developers. Some of these companies used to provide a list of full user reviews of a number of top mobile apps, such as 1000, to customers who were willing to pay a substantial subscription fee, such as 15000 – 30000 USD for a year, before the GDPR law came into force in the European Union on 25th May 2018. It was not clear to what extent the granularity of information was that those companies provided before spring 2018. Anyhow, these companies' current services have shrunk to typically App Analytics, App Marketing, App Store Optimization, and User Acquisition [8].

Many software projects have done great work on the analysis of social media. Some of them produce outstanding results in sentiment retrieval and real-time nature. However, projects in user review analysis are not often successful, especially in request elicitation and opinion

extraction. The main techniques in those projects are artificial intelligence (AI). For researchers, a well-known difficulty of AI is that it is difficult to adjust and control internal details. Researchers' adjustment and control of AI technology sometimes produce domain dependence or over-fitting. Therefore, the application of these technologies is difficult to achieve an excellent efficiency between computing resources and improved performance.

Based on the above considerations, this thesis proposes a framework solving user review analysis problems for the purpose of requirement elicitation, using non-machine learning techniques and using manually programmed linguistic rules, thereby achieving internal detail control of the system and avoiding topic domain dependency and over-fitting problems.

The hypothesis of this thesis is that such a framework that sets up a channel from raw user reviews to structured analysis data will be practical and usable.

The main reasons why this framework is believed to be able to solve above problems are:

- (1) The structure of this framework is composed of several loosely integrated components, which not only realize the flow of data from downloading raw user reviews to the structured analysis results, but also provide adaptability and flexibility for wider future applications;
- (2) The reasonable use of linguistic rules makes it possible to adjust and control the internal details of the system in this data flow;
- (3) Natural language processing (NLP) technologies, such as chunking, regular expressions, and especially Stanford dependency trees, provide substantial technical support for this framework.

This framework has been developed in the United Kingdom, where English is the most commonly spoken language. The framework in this thesis is designed to process user reviews in English in the current technical detail. A prototype has been implemented and tested on the user reviews data downloaded from 54 mobile apps on the Google Play platform. The framework will be provided as a starting point for other researchers to use or develop further for their own research questions.

Two sets of linguistic rules that are necessary for this framework to work have been implemented. One of those sets of rules is intended to identify user requests, and the other

identifies users' expressions of opinions about aspects of the mobile apps. Both sets of rules are manually programmed, rather than learnt by the system.

The performance of this prototype running on the three datasets proves the functionalities that could be good enough for other researchers to use this framework as a tool to carry out their own research.

1.2 Thesis statement and objectives

The aim of this PhD is to develop a framework that reads user review data and extracts meaningful information to support stakeholders in improving their products by satisfying user requirements.

Following this aim, a set of objectives are devised as follows:

- (1) To automatically extract app reviews from a major app store platform
- (2) To analyse the review data in order to produce meaningful information
- (3) To research and develop a method to evaluate the produced information
- (4) To validate the framework using examples from two or three mobile app user reviews datasets

1.3 The original contributions

For the research topic of user review analysis, the author implemented a light-weight solution that uses manually programmed linguistic rules and presents functionalities that could reach a good level of performance.

The contributions of this research are the following points:

- This framework is proposed to solve the user review analysis problem for requirement elicitation;
- The prototype of this framework proves its feasibility;
- The experiments prove the effectiveness and efficiency of this framework.

1.4 Legal considerations for downloading reviews and their publishing

Regarding whether this user reviews downloading activity is legal, and whether publishing the resulting data is legal, a process for the legal considerations reasoning was conducted. Issues such as copyrights, personal data, and legality conforming to the latest relevant laws were investigated.

The mobile app platform being used in this research is Google Play, which is known as an American company that provides active content in the United Kingdom (and the EU). There is a general agreement that the UK (and the EU) has stricter privacy laws than the US. Given that the research that is documented in this thesis is conducted in the UK, an initial assumption has been made that both this research and the Google Play content will comply with the laws active in the UK.

1.4.1 Copyrights and the Copyright, Designs and Patents Act 1988

The principal legislation on copyright can be found in the Copyright, Designs and Patents Act 1988 [9].

(1) Who owns the copyrights and how long are the copyrights' durations?

The authors of the reviews own the copyrights and their ownerships last approximately 70 years. Online reviews can be classified as original literary work, which is under the protection of Copyright Designs and Patents Act 1988 according to clause 1 (1) (a) of the Act [9]. According to clause 11 (1) of the Act, the first ownerships of the reviews belong to the authors of the user reviews.

Inducted from clause 12 subsections (1) to (6) of the Act, the durations of the copyrights of user reviews expire after 70 years from the data that they are first available to the public, or to which the user reviews are entitled in the countries of origin if not an EEA state, whichever comes first.

(2) Does the author of this thesis have the right to copy the user reviews?

The answer to this question is “Yes”. According to clause 29A subsection (1) of the Act, the copyright is not infringed if the copying is to “carry out a computational analysis” “for the

sole purpose of research for a non-commercial purpose” by a person who has lawful access to the work.

It is understood that because the user reviews are made available to the public already, the author has lawful access to the work. The author cannot reasonably provide an acknowledgement of the authors of the reviews due to data protection. The author will make copies of the lawfully accessible reviews for use in this non-commercial research.

Since the author’s activity to copy the user reviews in this research does not infringe the copyrights, any terms of contract to prevent or restrict this research activity are not enforceable, according to clause 29 subsection (4B) of the Act. This should apply to the terms and policies of any public platforms of user reviews.

(3) Does the author of this thesis have the right to transfer the copied user reviews?

The answer to this question is “No”. According to clause 29A subsection (2) of the Act, the copyrights will be infringed if the copied user reviews are transferred to any other person, except “authorised by the copyright owner”. Because the ownerships of the reviews remain with the authors, and it is not possible for the author of this thesis to obtain authorization from the review authors, the author of this thesis does not have the right to transfer the copied user reviews to any other person.

Moreover, according to clause 29A subsection (1), the computational analysis should be for “the sole purpose of research for a non-commercial purpose”. Therefore, the author will not use the copies of the reviews for anything other than this PhD non-commercial research.

1.4.2 Personal data, GDPR and the Data Protection Act 2018

The principal pieces of legislation on personal data that are currently active in the UK are the General Data Protection Regulation (GDPR) [10] and the Data Protection Act 2018 [11]. The Data Protection Act 2018 is the UK’s implementation of the GDPR.

(1) What is personal data and are user reviews personal data?

Personal data means “any information relating to an identifiable person who can be directly or indirectly identified in particular by reference to an identifier” [12]. The user reviews that are gathered in this research have names (although some are pseudonyms, such as “A Google

user”), review texts (users’ opinions of the app), and other data that arguably relate to the identifiable individuals. As such it is reasonable to conclude that the user reviews on the Google Play platform can be considered as personal data.

(2) Does the author of this thesis have the right to process the user reviews as personal data?

In Article 6 (1) of GDPR, it is defined that personal data processing shall be lawful if one of the six points applies. Point (f) defines that the processing is lawful if:

- Processing is necessary;
- For the purposes of the legitimate interests pursued by the controller or by a third party;
- The legitimate interests are not overridden by the interests, fundamental rights and freedoms of the data subjects.

This point applies to this research because:

- There is a legitimate interest behind this process, which is to inform a PhD research project
- The methodology relies on the processing of the data, therefore the processing is necessary
- There is no impact on the identifiable individuals, and the processing does not prevent them from exercising their rights

Thus, the author of this thesis can process the user reviews lawfully.

(3) What rights of the data subjects does the author of this thesis have to take into account in the processing?

Chapter three of GDPR defines the rights of the data subjects in great detail. In short, they are rights to breach notification, to access, to rectification, to be forgotten, to restrict processing, to data portability, to object, and the rights in relation to automated decision making and profiling.

The Google Play platform will deal with most of the rights by either reaching agreements with users when they post the reviews, or by website terms and policies. The only right that

the author of this thesis needs to take special care in addition to the others is the right to be forgotten, also known as data erasure. If the author permanently stores and shares the user reviews as part of the dataset, this obligation to the data subjects will not be fulfilled.

Therefore, the author of this thesis would be balancing the review authors' interests against the legitimate interest of the research by using the personal data of the reviews as sample data to create and test the techniques, but would then delete all copies of the reviews that the author holds and would not include any of the reviews in this PhD or accompanying files or data.

1.4.3 Conclusions from the legal considerations

The conclusions of the legal reasoning process are below:

- The copyrights of the reviews rest with the authors of the reviews
- User reviews are personal data that are protected by GDPR
- The author can copy the user reviews in this research for the non-commercial purpose
- The author can process the user reviews for this PhD research project lawfully
- The author cannot transfer the copied user reviews to any other person, or use them for another research project
- The author cannot permanently store and share the user review dataset, and will delete all copies of the reviews after this PhD

Therefore, this framework's deliverables will not contain any author names, full review texts or any identifiable personal data. But the research data such as the database definition, source code and the ontology definition will be included, especially along with the web sourcing code and instructions on how to gather review data, which is introduced in section 3.2.

1.5 Top level structure

This study proposes a framework for analysing online reviews in order to provide effective references for stakeholders. Taking the specific topic of mobile app user review as an example and focus, a simple and lightweight solution using linguistic rules has been implemented and proved its preliminary functionality with a good level of performance.

The overall structure after integration is a Batch Sequential style Data Flow architecture. This is shown in the figure below.

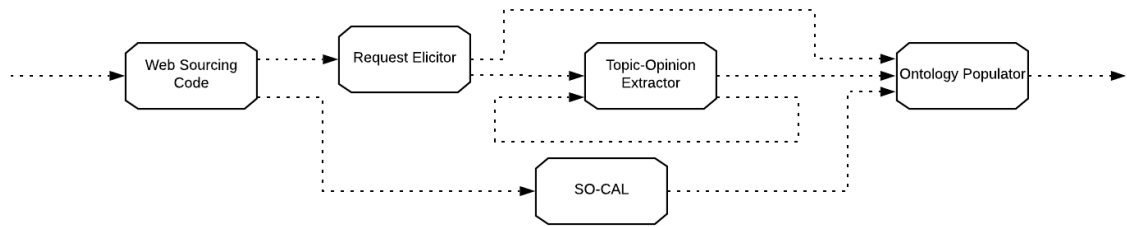


Figure 1 The framework architecture: Batch Sequential Data Flow.

Apart from the above system architectural diagram, this system can also be understood logically in an example that is illustrated on how a review is processed in the data flow in the figure below.

The framework starts by downloading user review data from the Internet as shown on the left in Figure 2. In practice, a set of web sourcing code was written for Google Play user reviews. When user reviews have been stored locally in a database, they are processed by the framework. Firstly, an adapted version of Semantic Orientation CALculator (SO-CAL) conducts sentiment orientation analysis and calculates the sentiment scores for the reviews. Secondly, the reviews are checked to establish whether they contain user requests (new requirements). If there is a user request, it is elicited by the Request Elicitor component. In this prototype, the preliminary user requests that are elicited are single pieces of information, namely whole phrases or parts of the sentences. Then the reviews are processed by a Topic-Opinion Extractor to extract all opinion and topic pairs (existing requirements) that relate to specific aspects of the mobile apps. In this prototype, the preliminary topics and opinions are single words at the present.

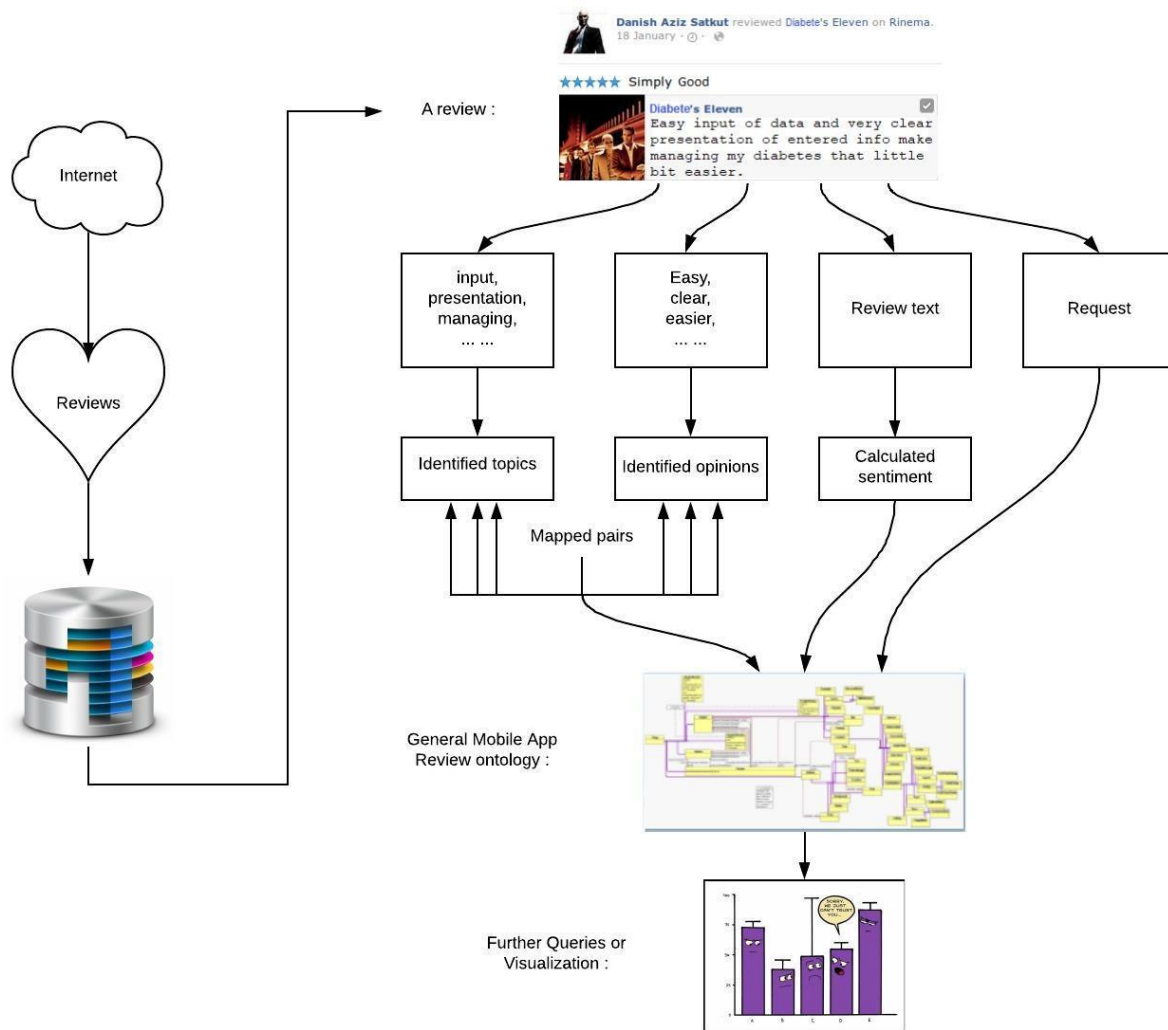


Figure 2 An example in the top level logic of this framework

When above processes have been done, namely the sentiment scores, the user requests, and all topics and opinions pairs, are ready, they are populated into the General Mobile App User Review Ontology together. These concept individuals and their relationships form a tree in the ontology. A series of Snap SPARQL queries are also provided in this framework to answer a few common example questions for users. Future users of this framework can derive their own SPARQL queries for their specific questions. Furthermore, they can output the data from the ontology to process further in order to get the data visualized or statistically analysed, such as in a spreadsheet software or a statistical tool, as depicted in the last step of Figure 2. Because the data formats or presentations are not always the same in the answers

of different questions, it is difficult for this prototype to fix a solution for the last step at the current stage. Figure 2 depicts the above processes with an example in the top-level logic of this framework.

1.6 Research methodology

The main research methodology in this work is to research and therefore use some current techniques to build up a framework, which is different with writing from a plain paper. A large amount of efforts has been invested into the research of available tools.

The dominant user review analysis technology today is machine learning. However, in order to avoid the topic domain dependency and overfitting problems in the performance, this framework attempts to use non-machine learning methods for user review analysis for the purpose of requirement elicitation. In order to achieve from the original user reviews to structured or semi-structured requirements data, this framework uses five components that are loosely connected. Such a structure not only achieves the desired data flow, but also avoids the encapsulated program's impact on the data volume that can be processed, and reserves space for future modifications and adaptability to new domains.

During the process that informed this framework, a series of conversations with domain experts took place. These discussions covered the technical focuses, architectural characteristics, and the scientific value it can contribute to forming such a framework. During the analysis of the preliminary experimental results, some further discussions also took place. These discussions further focus on the correctness of the user review classification results, the room for improvement, and the performance limits of such expert systems. These discussions are helpful in forming this framework.

In the development process of this prototype, each component is developed separately and tested independently. The structural characteristics of this framework determine that the final result can only be generated after all components have been developed and all linguistic rules for testing have been manually coded. This is a long process, and I thank my supervisor and Director of Studies for the patience he has given me along the way. A set of debugging code is also used to assist in the development of two key components, the request elicitor and the topic-opinion extractor, and the manual coding of linguistic rules. The linguistic rules of the

request elicitor balance the number of rules and the coverage of the rules. The linguistic rules of the topic-opinion extractor adopt the Open-Closed Principle in order to realize the convenient management of a relatively large number of rules while the linguistic rules accurately locate the data.

1.7 Thesis structure

The next sections are literature review, methods, results, discussion, and conclusion.

The literature review section introduces the states of the arts in the related area of this framework through some of the work that has been done by other researchers. The main focus is on opinion mining. It also mentions NLP in requirement elicitation and concept mapping onto ontologies.

The methods section presents the proposed approach first and then introduces the development methods for all five components in turn and some issues during their reuses.

The section of results and evaluation demonstrates results and evaluation of this framework after running on three datasets from Google Play. It also interprets the results and predicts what the best performance could be. Some Snap SPARQL queries are supplied to users for the initial uses of this GMAURO ontology.

The discussion section mainly discusses this framework's adaptability, flexibility, advantages and disadvantages for components, and some suggested further research directions from this prototype.

The conclusion section calls an end to this thesis.

2 Literature review

This chapter provides a brief background to the techniques that are used in user review analysis for requirement elicitation.

The main technique for user review analysis is opinion mining, which is introduced in section 2.1. The story starts from an opinion mining model that is described from a paper in 2004 [13]. In the context of this model, researchers were striving to extract topics and opinions separately, and bridge them later. But a number of researchers extract topics and opinions together, although they rarely produce the precise mapping relationships between topics and opinions for practical usages. This section relates to the technical background of the topic-opinion extractor component in this framework.

Sentiment orientation analysis and fake reviews are also briefly introduced in this section. Sentiment orientation analysis is achieved through component SO-CAL in this framework. As no research claims that fake reviews have reached a threshold to disable the validity of user review analysis, this framework and other research on user review analysis still remain useful.

How natural language processing techniques have contributed to requirement elicitation is introduced in section 2.2. In the later part of the section, a number of research on mobile app store user reviews for opinion mining and requirement elicitation is also introduced. A research paper that is similar to the topic-opinion extractor component is also reported in this part.

The last section 2.3 presents the very typical research on concept mapping onto ontologies. This relates to how the prototype maps the information from the previous components onto the general mobile app user review ontology in this thesis.

2.1 Opinion mining

Finding the opinions within the reviews and classifying them are also referred to as sentiment analysis, sentiment classification, or opinion mining. However, along with the techniques

involved and their applications, the categories of techniques and applications derived into a considerable number.

For example, based on the documents, there are document level, sentence level, phrase level and word level. Based on the aspects, there are opinion aspect, target (topic) aspect, and both. Based on the techniques, there are lexicon based, machine learning based, natural language processing (NLP) based, and hybrid. Based on the specific machine learning or AI techniques, there are supervised, semi-supervised, unsupervised, which could be further combined with different models such as topic modelling, bootstrapping, rule based, graph based, syntactic tree based. Based on the information, there are explicit and implicit. Based on the applications, there are subjectivity classification, semantic orientation classification, spam detection, and cross domain sentiment analysis. Based on the media, there are text based and multimodal based. In addition, these above categories can be further combined.

It is not surprising that the literature on sentiment analysis is so huge that researchers do not reach the same train of thoughts when writing surveys [14]–[23]. In order to avoid biases and incomprehensiveness, some researchers tend to focus on one aspect only in their surveys. Table 1 lists ten surveys from different focuses.

Table 1 Different focuses of surveys in sentiment analysis

Year	Paper	Focus of the Survey
2010	Sentiment analysis [14]	General introduction for sentiment analysis, evaluation and applications
2010	A Survey on the Role of Negation in Sentiment Analysis [15]	Negation in sentiment analysis
2015	A survey on opinion mining and sentiment analysis: Tasks, approaches and applications [16]	Statistic over the techniques used and the applications of the 161 papers

2016	Aspect extraction in sentiment analysis: comparative analysis and survey [17]	Aspect-Level Analysis	Sentiment
2016	Survey on Aspect-Level Sentiment Analysis [18]	Aspect-Level Analysis	Sentiment
2017	A review of natural language processing techniques for opinion mining systems [19]	The usages of NLP techniques	
2017	A survey of multimodal sentiment analysis [20]	Multimodal sentiment analysis	
2017	A systematic literature review: Opinion mining studies from mobile app store user reviews [21]	Sentiment analysis in mobile app store user reviews area	
2018	Extracting useful software development information from mobile application reviews: A survey of intelligent mining techniques and tools [22]	Sentiment analysis in mobile app store user reviews using intelligent techniques and tools	
2018	Distinguishing between facts and opinions for sentiment analysis: Survey and challenges [23]	Subjectivity detection	

Sentiment analysis [14]

Zhao, Qin and Liu made general introductions on sentiment analysis, evaluation and applications. For the techniques and methods used in sentiment analysis, including sentiment extraction, classification and summarization, they gave relatively comprehensive explanations up until the time of their writing. The survey is in Chinese with an English abstract.

A Survey on the Role of Negation in Sentiment Analysis [15]:

Wiegand *et al.*'s survey on negation in sentiment analysis is comprehensive not only at the time of writing. Their insights in the negation models, approaches and challenges are still valid at present. In their motivation section, they introduced thoroughly different negation

scenarios. In the survey section, they introduced and explained the main research directions and their advantages and disadvantages. Negation includes not only negation words, but also other units, such as diminishers, negation within words, and irony. A good point they also made is that modelling the scope of negation expressions is useful.

The negation treatment in this prototype submitted with this thesis simply focuses on explicit negation words, not diminishers, nor irony. The reason for this simple approach to have considerable acceptable performance is the use of Stanford dependency trees. The Stanford parser parses English sentences into well-structured trees. The prototype only considers negation words appearing as the parent, a sibling or a child of the opinion word. In the resulting performance, negation error does not contribute to the error reasons as an obvious reason at all. However, how to further improve the negation handling in this prototype is still an interesting question.

It is not necessary to expand the contents in each of the above surveys. The readers are referred to above specific surveys for more details.

2.1.1 A traditional opinion mining model

Considering the work involved in this thesis, the focus of this subsection will be an introduction of a traditional model for opinion mining and the story that researchers use this model to mine opinions, targets (topics) and bridging them together.

Although researchers chose different aspects and approaches to present their understandings in their surveys and findings, a typical popular model is presented in Munezero *et al.*'s paper [24]. In this model, researchers think a sentiment is composed of a sentiment holder, an emotional disposition, and an object.

It is not clear who is the original author of this model because it could be traced back to 2004 from the paper by Kim and Hovy [13], who tried to identify those elements and combine them together, and actually described an opinion as a quadruple [Topic, Holder, Claim, Sentiment]. It could be arguable whether this model should be credited to Kim and Hovy although they did not name it. Alternatively, researchers might prefer to think that this is a common English structure.

An example of an opinion that conforms to this model is in Figure 3.

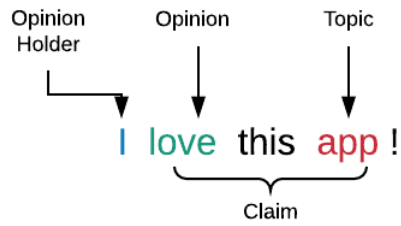


Figure 3 Example of an opinion with an opinion holder

However, it frequently occurs that there is no opinion holder in a review sentence. This is a very common scenario in mobile app user reviews, e.g. the example depicted in Figure 4.

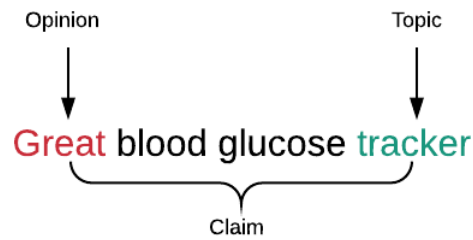


Figure 4 Example of an opinion without an opinion holder

Figure 4 precisely describes the task that the Topic-Opinion Extractor component of this prototype currently can do: extracting pairs of topic words and opinion words.

Many researchers had a similar train of thoughts to Munezero *et al.* [24]. Researchers did a lot of research on finding the correct emotional dispositions or refinement of the opinion vocabularies [25]–[32].

Table 2 Research of mining or refining opinions

Year	Paper	Methods of mining opinions
1997	Predicting the semantic orientation of adjectives [25]	Corpora based, classification of adjectives' orientation

2000	Learning Subjective Adjectives from Corpora [26]	Corpora based, clustering adjectives words according to distributional similarity
2003	Measuring Praise and Criticism: Inference of Semantic Orientation from Association [27]	Corpora based, inferring the semantic orientation of a word from its statistical association with a set of positive and negative paradigm words
2003	Learning extraction patterns for subjective expressions [28]	Corpora based, bootstrapping process fed with syntactic patterns
2005	Automatic Detection of Opinion Bearing Words and Sentences [29]	Lexicon based, from opinion words mining to sentences subjectivity classification
2006	Mining WordNet for fuzzy sentiment: Sentiment tag extraction from WordNet glosses [30]	Lexicon based, analyse the WordNet through a three-step (pass) method against 1904 adjective seeds, to form a sentiment strength distribution of WordNet terms
2009	Subjectivity recognition on word senses via semi-supervised mincuts [31]	Lexicon based, supplement WordNet entries with information on the subjectivity of its word senses through a semi-supervised approach
2012	Automatic detection of political opinions in tweets [32]	NLP techniques applied in their GATE tool

While some researchers contributed more when looking for the objects [33]–[38].

Table 3 Research contributed to mining opinion targets (topics)

Year	Paper	Methods of identifying topics
2008	Modeling online reviews with multi-grain topic models [33]	based on extensions to standard topic modelling methods such as LDA and PLSA to induce multi-grain topics; not only extract aspects, but also cluster them into coherent topics
2008	Topic identification for fine-grained opinion analysis [34]	Proposed an algorithm for opinion topic identification, but this paper only focuses on the first half: clustering opinions that share same topics, which was achieved by adapting a standard machine learning-based approach to noun phrase co-reference resolution (the next step will be labelling the clusters with the name of the topic)
2011	Opinion word expansion and target extraction through double propagation [35]	Double propagation; Rule-based; Minipar was used in this paper. (At the time of writing this thesis, the Minipar homepage shows 404 Not Found messages.)
2012	System and method for automatically summarizing fine-grained opinions in digital text [36]	This patent continues the research in their 2008 paper above [34]. For the topic identification, they use predefined topic lists to train topic models classifiers with supervisory labels. If a sentence contains an opinion expression, the classifiers determine which of the topic categories are represented in the sentence.

2017	RubE : Rule-based methods for extracting product features from online consumer reviews [37]	<p>Rule-based unsupervised methods that extract both subjective and objective features;</p> <p>Subjective features by extending double propagation with indirect dependency and comparative construction</p> <p>Objective features: by incorporating part-whole relation and review-specific patterns</p>
------	---	---

2017	A two-fold rule-based model for aspect extraction [38]	<ol style="list-style-type: none"> 1) Using sequential patterns-based rules to extract explicit aspects that are associated with regular opinions; 2) Improving aspect extraction accuracy with a frequency-based approach along with normalized Google distance; 3) Extracting aspects that are associated with domain dependent opinions.
------	--	--

Moreover, some researchers made contributions on identification of the sentiment holders when dealing with sentiment analysis [13], [39]–[44].

Table 4 Research contributed to identifying opinion holders

Year	Paper	Methods of identifying opinion holders
2004	Automatic extraction of opinion propositions and their holders [39]	<p>Bethard <i>et al.</i> labelled holders when they labelled propositional opinions. They labelled all agents of opinion-propositions as holders, but counted implicit holders that were “speakers” or un-lexicalized as no holders.</p>

2004	Determining the sentiment of opinions [13]	Kim and Hovy used a named entity tagger to identify potential opinion holders. Only “Person” and “Organization” that are closest to the topic phrase are considered.
2005	Identifying sources of opinions with conditional random fields and extraction patterns [40]	An automatic supervised learner is created to extract opinion sources. The opinion sources that Choi <i>et al.</i> defined are much broader than the opinion holder concept above. Choi <i>et al.</i> consider “authority”, “location”, and “proper name” as opinion sources. In their work, one opinion source is identified in one sentence.
2006	Identifying and analysing judgment opinions [41]	Kim and Hovy define a holder as one of the elements of a judgement opinion. They use Charniak parser to interpret sentence tree structures. Based on the paths on the trees, the type of the holder candidates, and the distance between the holder and the opinion expression, their learner is able to identify multiple holders for multiple opinions in one sentence.
2006	Extracting opinions, opinion holders, and topics expressed in online news media text [42]	<ul style="list-style-type: none"> (1) Identify opinions (2) Label semantic roles related to the opinions (3) Find holders and topics of opinions among the identified semantic roles (manually built a mapping table to map elements to holder or topic)

2010	Convolution kernels for opinion holder extraction [43]	Different convolution kernels are investigated for opinion holder extraction. The performance is the best if all kernels are combined. Stanford Parser is used.
2014	Frame-based detection of opinion holders and topics: a model and a tool [44]	A semantic representation opinion model of sentences is proposed in this paper. Opinion holders are annotated to the “agent” that is associated with four types of “opinion trigger verbs”.

Lastly, researchers bridged them together at some points [45], [46].

Table 5 Research of bridging topics and opinions together

Year	Paper	Methods of association
2003	Sentiment analyzer: extracting sentiments about a given topic using natural language processing techniques [45]	1) a topic specific feature term extraction, 2) sentiment extraction, and 3) (subject, sentiment) association by relationship analysis
2007	Extracting appraisal expressions [46]	An attitude type taxonomy and two domain-dependent target type taxonomies are predefined. Stanford Parser is used for dependency representation. A ranked list of linkage specifications of the paths in the dependency trees is hand-constructed. When linking each attitude to a target, the link between them is compared with the ranked list and prioritized.

Sentiment analyzer: extracting sentiments about a given topic using natural language processing techniques [45]:

Yi *et al.* extract candidate feature terms based on a set of Part-Of-Speech (POS) patterns first, then select them through two algorithms: a mixture language model and likelihood ratio [45].

For sentiment phrases, they use a syntactic parser to identify subject, object, adjective, and prepositional phrases of a sentence. From those elements, they identify all sentiment adjectives as sentiments. If the feature terms identified before have sentiment words, they become sentiments.

For the feature / target and sentiment association, they use a sentiment pattern database to determine a sentiment phrase's target and polarity, if the target exists in the sentence. For sentences that do not match the sentiment pattern database, they assign the association based on a group of B-expressions (for incomplete fragments containing sentiment) if possible.

This is a typical early work that researchers identify sentiments and targets/topics separately, and then bridge them together.

Because Yi et al.'s datasets are selected with a subject term in each sentence, and they are datasets from 15 years ago, their reported performance can not act as baselines to compare with the evaluation datasets used for the prototype in this thesis. In the evaluation of this thesis, the reviews are randomly sampled without any restriction or selection process.

Extracting appraisal expressions [46]:

Bloom, Garg and Argamon constructed an attitude type taxonomy and two domain-dependent target type taxonomies. Both are in tree structures and have vocabularies as leaves under each category.

They then use Stanford Dependency Parser for dependency representation. They hand-constructed a ranked list of linkage specifications of the paths in the dependency trees that connect attitudes and targets. Priorities are manually assigned for the paths so only the highest priority specification can be used when more than one paths are found. After attitudes and target candidates are found, in order to link each attitude to a target, the links between them are compared with the ranked list and prioritized.

Worth noting, if no linkage is found at all, they assign the default category of targets to the attitudes. This situation frequently happens in user reviews data used in this thesis. This thesis adopts a similar method to their approach: assigning the opinions to the current mobile app.

Bloom, Garg and Argamon used Stanford Dependency Parser in their work. Although they adopted the similar approach as other researchers at that time, and used Stanford dependency trees as mainly the linking methods to bridge the previously resulting attitudes and targets, it was a significant sign that Stanford dependency parser began to contribute to research considerably.

It could be arguable whether these jobs could be easier if with more help of the Stanford Dependency Parser [47]. However, the Stanford Parser might not have reached sound robustness and gained enough popularity at the time of this paper was written, despite that its first release was in 2002 [48].

2.1.2 Co-extraction

Continuing the story in the previous subsection, along with the techniques' development, some researchers adopt a different approach: mining the opinions and targets together or helping each other's mining. This approach is named co-extraction.

Co-extraction papers are few. A number of papers that aim to extract topics and opinions at the same time are reviewed. The results of the review yielded nine papers that feature co-extraction as listed in Table 6.

Table 6 Papers surveyed for topics and opinions co-extraction

Year	Paper	Methods of co-extraction
2004	Mining Opinion Features in Customer Reviews [49]	A frequent feature: nearby adjective → opinion A non-frequent feature: nearest noun or noun phrase → feature of the nearby opinion

2005	Extracting Product Features and Opinions from Reviews [50]	Syntactic dependencies in vicinity computed by the Minipar parser embodied by 10 extraction rules
2010	Jointly Modeling Aspects and Opinions with a MaxEnt-LDA Hybrid [51]	<p>MaxEnt-LDA that uses a modified version of the multi-grain topic models [33]</p> <p>Feature: Assume one sentence has a single feature</p> <p>Opinion: a general opinion model and a number of aspect-specific opinion models</p>
2012	Cross-Domain Co-Extraction of Sentiment and Topic Lexicons [52]	<p>Use labelled data from a source domain to help the target domain:</p> <ol style="list-style-type: none"> (1) generate opinion seeds in target domain if they are common in both source and target domains (2) extract syntactic relationships among topics and opinions through dependency trees (3) extract topic seeds in target domain through the relationships identified in step 2 (4) expand both topic and opinion seeds in target domain in a new bootstrapping-based method
2014	Extracting Opinion Targets and Opinion Words from Online Reviews with Graph Co-ranking [53]	<p>Graph co-ranking over three concepts: semantic relations, opinion relations, and word preference</p> <p>Opinion relations are relationships between topics and opinions, which is achieved through</p>

			word alignment combined with partially supervised syntactic information.
2015	Co-Extracting Targets and Opinion Words from Online Reviews Based on the Word Alignment Model [54]	Opinion	<p>(1) partially supervised word alignment model → identify opinion relations</p> <p>(2) graph-based co-ranking → estimate the confidence of each candidate</p> <p>(3) extract topics and opinions that have higher confidence</p>
2016	Aspect-based Sentiment analysis using a stacked neural network architecture [55]	relational	<p>This paper addresses three tasks: identification of aspect and opinion terms; labelling opinions with sentiments; and extracting relations between opinions and aspects.</p> <p>For the relation extraction, they use a corpus that contains relation labels to predict many-to-many relations between aspect and opinion terms with the help of a recurrent neural network.</p>
2017	Coupled Attentions for Co-Extraction of Aspect and Opinion Terms [56]	Multi-Layer	<p>A deep learning model: coupled multi-layer attentions</p> <p>It does not require any parsers or other linguistic resources for pre-processing. But it needs either pre-trained word embedding or manual labelling to form the training sets.</p>
2019	Global Inference for Aspect and Opinion Terms Co-Extraction Based on Multi-Task Neural Networks [57]		<p>(1) a multi-task learning framework based on neural networks</p> <p>(2) a global inference method through several syntactic constraints, which are based on the Stanford Dependency Parser</p>

Mining Opinion Features in Customer Reviews [49]:

Hu and Liu investigate the method of co-extraction in this paper of 2004. They extract frequent features first and prune them to a satisfactory level. Then use the features to extract opinions. Later, the opinions help to extract infrequent features. Finally, the semantic orientations of opinions are identified, and the sentence's semantic orientation is decided through the dominant orientation of the opinion words.

Worth noting, the method they use to mine associated opinions with topics is rather basic compared with the state of the art today. They used the nearby neighbours. When looking for opinions, if a sentence contains a frequent feature, they extract the nearby adjective as an opinion. When looking for infrequent features, if a sentence contains an opinion, they extract the nearest noun or noun phrase as an infrequent feature.

Although the method has been abandoned nowadays, this may be the first paper that made the efforts to extract topics and opinions together when considering the associations of them.

Extracting Product Features and Opinions from Reviews [50]:

Popescu and Etzioni also think a product feature's associated opinion will occur in its vicinity. However, they use Minipar parser to compute the syntactic dependencies instead. 10 extraction rules were used in the extraction. If a sentence has an explicit feature, the rules look for the heads of the opinion phrases. Then each head and its modifiers are returned together as a potential opinion phrase. But only the opinion phrases whose heads are positive or negative are retained as opinions.

This paper can be regarded as iconic in co-extraction papers for the attempt to investigate syntactic dependencies between features and opinions based on rules.

Jointly Modeling Aspects and Opinions with a MaxEnt-LDA Hybrid [51]:

Zhao *et al.* model a MaxEnt-LDA that uses a modified version of the multi-grain topic models [33]. Their LDA model captures both topics and opinions without looking into the relationships between them.

They assume there are T aspects of topics in the given reviews. One sentence is assigned to a single aspect only. They further divide opinions into two types: general opinions and aspect-

specific opinions. Then a set of labelled training sentences were used to train the Maximum Entropy model. The trained data were then used to infer on other data.

The results are able to show relatively general corresponding relations between topic aspects and aspect-specific opinions, namely, which opinion relates to which aspect. This is an important attempt towards the direction of providing precise mapping relationships between topics and opinions. The authors also make their ambition clear in the conclusion section that they plan to use this model to help sentence-level extraction of specific opinions and their targets.

It is necessary to claim here that this research that is reported in this thesis accomplishes Zhao et al.'s ambition. The prototype in this thesis not only extracts topics and opinions in a sentence, but also keeps their mapping relationships that are traceable, regardless they are general opinions or aspect-specific opinions. Moreover, this prototype is able to extract multiple topic-opinion pairs in a sentence, and there is no restriction on how many aspects there are in the sentence.

Cross-Domain Co-Extraction of Sentiment and Topic Lexicons [52]:

Li *et al.* have data from two domains: a source domain and a target domain. The source domain has plenty of labelled data that are used to help the extraction in the target domain. The method can be summarised into steps below:

- (1) To generate opinion seeds in the target domain if they are common in both the source and the target domains
- (2) To extract syntactic relationships between topics and opinions through dependency trees (if they are precise in the source domain and frequently appear in the target domain)
- (3) To extract topic seeds in the target domain via the relationships identified in step 2
- (4) To expand both topic and opinion seeds in the target domain in a new bootstrapping-based method (in an iteration process, a cross-domain classifier is used to predict the unlabelled data in the target domain. Then a word graph is built between the topics and opinions according to the up-to-date extracted syntactic relationships. After that,

a score refinement algorithm is performed on the graph, in order to extract new words with the top scores.)

This is an interesting paper that also uses syntactic relationships between topics and opinions. Specifically, the concept of dependency trees clearly appears four times in this paper. The authors also have the similar idea with previous researchers [49], [50] so that they used the shortest paths as the relationships between topics and opinions. It would be great if they have also documented which parser they used to decide the relationships or what method they adopted to produce the dependency trees.

Extracting Opinion Targets and Opinion Words from Online Reviews with Graph Co-ranking [53]:

Liu, Xu and Zhao proposed a graph co-ranking approach to extract topics and opinions together. They introduce three concepts: semantic relations, opinion relations, and word preference.

Opinion relations are relationships between topics and opinions, which is achieved through word alignment combined with partially supervised syntactic information.

Semantic relation is a new concept that is used to group similar topics together. This could partially be a similar concept to the “keywords” two-dimensional array that is used in the prototype presented in this thesis. The reason they introduce this concept is their curiosity to whether only the syntactic relationships between topics and opinions are sufficient for co-extraction. This is a valid question.

Word preference is also a new concept that the authors introduced in this paper. They argue that certain words combinations have higher frequencies therefore should be considered with higher preferences. In their experiments, word preference did enhance the performance. However, it is arguable whether this is because their machine learning algorithm predicts the results with probabilities and word preferences actually help their word candidate confidence calculation.

The graph that Liu, Xu and Zhao have constructed in this paper is different with the graph that Li *et al.* has built [52]. Li *et al.* built a graph that only contains relationships between topics and opinions, which is the concept of opinion relations in this paper: subgraph G^{to} . In

this paper, Liu, Xu and Zhao built three subgraphs: G^{tt} , G^{oo} and G^{to} . G^{tt} is covered by the “keywords” list in the prototype submitted in this thesis. G^{oo} is the relationships among opinions. In this thesis submission, the GMAURO ontology automatically put the same opinions under one IRI, which could partially be similar with this subgraph.

Moreover, word preference is also fed into the co-ranking algorithm in this paper, which helps the performance when estimating the confidence of each word candidate.

Co-Extracting Opinion Targets and Opinion Words from Online Reviews Based on the Word Alignment Model [54]:

This paper identifies opinion relations in the first step through a partially supervised word alignment model. Secondly, a graph-based co-ranking algorithm is performed to estimate the confidence of each candidate. Finally, topics and opinions are extracted if they have higher confidence.

Liu, Xu and Zhao’s this paper gave more details of how the opinion relations are identified through the partially supervised word alignment model. When training the word alignment model, syntactic patterns are fed into the model. For English, they employ the Minipar. For Chinese, they use the Stanford Parser.

They regard some short or direct syntactic relations as precise, whereas indirect dependency relations decrease the precision. This is a common perception that is also shared in this thesis. Therefore, the syntactic patterns are based solely on the direct dependency relations in their research. The prototype in this thesis also does the same, namely, only considers direct relationships (single arcs on the dependency trees) between topics and opinions.

Aspect-based relational Sentiment analysis using a stacked neural network architecture [55]:

Jebbara and Cimiano propose a neural architecture that consists of a neural network based component for each of the three subtasks: identification of aspect and opinion terms; labelling each opinion term with a sentiment; extracting relations between the opinion and aspect terms.

In the first component for aspects and opinions extraction, a hybrid architecture consisting of a recurrent neural network stacked on top of a convolutional neural network is proposed.

In the second component to extract the sentiments of opinions, a recurrent architecture in combination with word distance features is proposed.

The third component that extracts relations between aspects and opinions uses labelled relation data in a corpus. A recurrent neural network is used to predict many-to-many relations between aspect and opinion terms. The network's output is interpreted as the probability that the pair of aspect and opinion terms form an aspect opinion relation.

Coupled Multi-Layer Attentions for Co-Extraction of Aspect and Opinion Terms [56]:

Wang *et al.* propose a deep learning model: coupled multi-layer attentions. For each sentence, they construct a pair of attentions, one for topics and one for opinions. To train the model for direct relations between topics and opinions, the pair of attentions are coupled in learning. To capture indirect relations among topics and opinions, they construct a network with multiple layers of coupled attentions.

The authors claim that the proposed model does not require any parsers or other linguistic resources for pre-processing. However, for the purpose of training, it still needs either pre-trained word embedding or manual labelling to form the training sets.

Global Inference for Aspect and Opinion Terms Co-Extraction Based on Multi-Task Neural Networks [57]:

Yu, Jiang and Xia first apply a multi-task learning framework based on neural networks, which implicitly captures the relations between topics and opinions. They then propose a global inference method to uncover the explicit relationships between topics and opinions and among themselves. The global inference is mainly through several syntactic constraints, which are based on the Stanford Dependency Parser.

The concepts that are used by Yu, Jiang and Xia are similar with the ones that are used by [53] in structure. The reasons that they did not compare together are possibly because of the different methods and datasets they used respectively.

Worth noting that Stanford Parser achieves best performance in most cases therefore is employed in this paper.

Common advantages of above co-extraction papers:

The common character of these co-extraction papers is that they all use machine learning or AI techniques. This has mainly two benefits below:

- (1) capability to support complex or indirect relationships between topics and opinions

Indirect dependency relationships are complex than direct relationships and their performance is not necessarily better.

- (2) capability to support phrases

Phrases help to provide wider contexts in opinion mining. The prototype in this thesis currently does not support phrase mining. This could be a further improvement direction.

Common disadvantages of above co-extraction papers:

The common limitations of these co-extraction papers are below:

- (1) Separated topics and opinions output results

Although syntactic relationships or labelled relation data between topics and opinions are fed into their models, the outputs are mainly one set of topics and one set of opinions, or probabilities of the relationships. The mapping relationships between each topic and each opinion are either missing from the output, or not reported.

- (2) Lack of explanations of how negation is treated

None of these papers reported how negation is treated. For example, the users have completely different opinions when they say, “this app is good” and “this app is not good”. The models have to adjust themselves to accommodate negations. Because negation is a common phenomenon in natural language, its treatment is inevitable in machine learning. However, it is understandable that considering negations, even only the negation words, will significantly make the models complex.

- (3) Repeating datasets

It is a common phenomenon that a paper reuses datasets from other papers or from sources that are publicly available. However, this could cause some problems.

The first problem is the possible performance changes if the datasets are replaced by up-to-date datasets from the real world. It is widely accepted that people speak in different language patterns and vocabularies along with the time changes. Publicly available datasets contain frozen language patterns and vocabularies that are tied to the time of the datasets were taken.

The second problem is the legality. In the UK, user reviews belong to the users themselves and they are personal data. It is illegal to source the data without the consent of the authors for commercial purposes. For non-commercial purpose research, it is illegal to transfer the personal data to any other person. Non-commercial researchers are also not allowed to keep the data permanently after the research is completed. This is discussed in detail in the subsection 1.4.

The third problem is the potential domain related performance or overfitting. This is potential because there is a possibility that certain datasets may persist some linguistic patterns. If there is anything that occurs in the dataset without being noticed or fully compared, the trends may exist in most of the research results that use them or even get exaggerated.

A little story line of dependency parsers in above co-extraction papers:

In these topics and opinions co-extraction papers, syntactic dependencies are first used by [50]. They used 10 rules through Minipar.

It was then mentioned again in 2012 [52]. However, it is not clear which parser the authors used in this paper.

In 2015 Liu, Xu and Zhao [54] explained in their paper that the word alignment model was partially supervised and fed with syntactic patterns. For English, they used Minipar, whereas for Chinese, they used Stanford Parser.

Until 2019, Yu, Jiang and Xia [57] fully utilized Stanford Parser for their whole research because of the best performance in most cases.

The Stanford dependency parser is one of the main techniques used in the prototype of this thesis. By recursively traversing each dependency tree of a sentence, this technique extracts precise matching relationships between each direct pair of topic and opinion, provided the tree parsing is correct. This method avoids a lot of work to match the pairs in other research scenarios if in that the dependency trees were not traversed. As long as the dependency tree

parsing performance is robust enough, it is considered that this is the best matching technique between topics and opinions. Each pair of topic and opinion is connected through a direct arc on the dependency tree. Only direct relations are considered in this prototype. The whole set of topics and opinions are likely to be covered through the whole datasets because users repeat similar topics and opinions in different reviews.

Compared with the previous discussions on researchers' work of finding sentiments and topics, then bridging them together, this PhD project avoids the hard work and inaccuracy, therefore serves as a helpful alternative by pairing each topic and opinion from the beginning of the extraction. This is the first published research that uses none machine learning techniques to extract precise topic-opinion pairs through dependency trees and keep the mapping relationships. In addition, it does not have serious problems that are typified to machine learning approaches such as domain dependency or overfitting.

2.1.3 Sentiment orientation analysis

Sentiment orientation analysis is also commonly referred to as sentiment polarity analysis or classification. Researchers usually divide subjective texts into two categories: positive and negative. The classification can be done on different text granularities, such as words, phrases, sentences and whole articles.

Kim and Hovy treat word sentiment classification into a positive, negative and neutral three-way classification problem instead of the previous two-way problem of positive and negative [41].

Generally speaking, there are two main approaches, machine learning and lexicon based [32]. The majority of the automated sentiment orientation research involves machine learning [58]. Other research has been lexicon-based [59] or a combination of the two [60], [61]. The combination of both approaches is typically with the lexicon-based approach being the key [62].

A number of researchers have combined above two methods to reach better performance. Dhaoui *et al.* [61] propose a combined approach that “greatly improved” positive sentiment classification without sacrificing the negative classification. While Kolchyna *et al.* [60] also

claim their combined approach produces more precise classifications. A handful papers [32], [55], [58]–[61], [63] that work on sentiment orientation analysis and are mentioned above are listed in Table 7.

Table 7 Some papers on sentiment orientation classification

Year	Paper	Methods of classification
2008	An ontology-based sentiment classification methodology for online consumer reviews [63]	ontology-based sentiment classification: a support vector machines sentiment classifier utilizes the lexical variations and synonyms in the ontology
2011	Lexicon-Based Methods for Sentiment Analysis [59]	This paper describes the design and implementation of the Semantic Orientation CALculator (SO-CAL), which is applied to the polarity classification task
2012	Automatic detection of political opinions in Tweets [32]	GATE is an architecture tool for users to incorporate certain NLP techniques and rules to handle linguistic problems.
2016	Aspect-based relational Sentiment analysis using a stacked neural network architecture [55]	This paper addresses three tasks: identification of aspect and opinion terms; labelling opinions with sentiments; and extracting relations between opinions and aspects. For the sentiment extraction, they propose a recurrent architecture in combination with word distance features.
2016	Annotate-Sample-Average (ASA): A New Distant	A distant supervision method that uses unlabelled tweets and prior opinion lexicons is

	Supervision Approach for proposed to generate synthetic training data for Twitter Sentiment Analysis [58]	twitter polarity classification.
--	---	----------------------------------

2016	Twitter Sentiment Analysis: Lexicon Method, Machine Learning Method and Their Combination [60]	This paper investigates lexicon based and machine learning approaches and their different combinations.
------	--	---

2017	Social media sentiment analysis: lexicon versus machine learning [61]	Consumer comments from Facebook brand pages are used in the experiments. Lexicon-based, machine learning approaches and their combination were compared. The combined approach significantly improved performance.
------	---	--

Lexicon-Based Methods for Sentiment Analysis [59]:

Taboada *et al.* are the authors of SO-CAL [64], which is adapted to one of the components of this prototype in this thesis. SO-CAL calculates the polarity of one or a number of documents and produces a sentiment value for each document. SO-CAL is lexicon based, namely it relies on a number of dictionaries. It handles not only the lexicons' polarities and strengths, but also intensifications and negations.

As introduced in the survey paper on negation in sentiment analysis [15], negation is a complex problem. SO-CAL makes great efforts in negation calculation, which helps the final polarity results considerably.

Furthermore, domain dependency is a challenge for app store user review analysis [21]. Cosma *et al.* present a system using a set of generalized grammar rules to overcome the domain dependency barrier in supervised opinion extraction machine learning [65].

Generally, the more specific domain related, the better performance an analysis gets. This study aims to eliminate domain limitations and provide a framework whose flexibility in terms of domains can reach a better level.

The framework proposed in this study is based on some mature library or tools, such as NLTK [66] and the Stanford Dependency Parser [47], and lexicon-based tools: SO-CAL [67] and WordNet [68].

Nevertheless, there is still potential to improve performance in the sense of separating domain terminologies from the GMAURO ontology, or mining reviews on the domain terms, if specific domain knowledge gets involved.

2.1.4 Fake reviews

Online reviews contain spam. Because users tend to rely on reviews, the trust also attracts fake reviews to artificially promote or devalue products. Although no researcher claims that fake reviews have reached a threshold that could lead to the whole online reviews becoming untrustworthy, researchers are constantly working on fake reviews detection. Table 8 lists some research in this area.

Table 8 A few typical research on fake reviews

Year	Paper	Methods of detection
2013	Fake Review Detection: Classification and Analysis of Real and Pseudo Reviews [1]	A method to distinguish fake reviews using KL-divergence and its asymmetric property is proposed. An additional set of interesting reviewer behavioural features also improves the classification result.
2014	Towards a General Rule for Identifying Deceptive Opinion Spam [5]	Li <i>et al.</i> adopt a generative Bayesian approach, in which a combination of topic models and generalized additive models are used. For intra-domain classification, they find unigram features constantly outperform general features.

		For cross-domain classification, they find using more general features to be more robust than unigram features alone.
2015	Survey of review spam detection using machine learning techniques [2]	Various machine learning techniques on detecting review spam and the performance of different approaches for classification and detection are reported in this survey.
2017	An effective hybrid Cuckoo Search with Harmony search for review spam detection [4]	In order to select an optimized feature subset for a dataset, a hybrid cuckoo with harmony search is proposed. Naïve Bayes is used for classification. The test results show that the chosen feature subsets provide better classification accuracy.
2018	An Attribute Enhanced Domain Adaptive Model for Cold-Start Spam Review Detection [3]	<p>You, Qian and Liu propose a deep learning architecture for incorporating entities and attributes from various domains into a unified framework. They present a domain classifier to adapt the knowledge across domains. Their research solves the data scarcity problem in the cold-start settings (new reviews from new reviewers).</p> <p>One of their interesting findings is that the date feature is critical in spam review detection. This potentially indicates that spammers may post reviews immediately after registering at the website.</p>

2.2 NLP in requirement elicitation

The aim of requirements elicitation is the acquisition from stakeholders of information of a system-to-be [69]. In requirement engineering, once requirements are set up, they are supported by various tools throughout the software life cycle. The techniques to automatically convert requirements between different models even into coding are developed.

However, the gap between the various forms of natural language requirements and the well-established models still remain open. Researchers have been working on this gap from late last century. Do Prado Leite and Franco develop a conceptual model, Language Extended Lexicon that aims to help knowledge modelling in requirements engineering [70]. Following this approach, a number of researchers proposed different requirement specification language standards. Among those standards, Semantics Of Business Vocabulary And Rules (SBVR) [71] seems to be accepted by more researchers for the reason that it not only generates software models but also is machine process-able.

In order to generate software models from natural languages, the languages have to follow the standards in requirement specifications. This is hard even for business analysts and requirement engineers, provided that agile methodology has gained popularity in the recent decades. When working with an agile methodology, people often sacrifice the quality of the requirement documentations for speed. In addition, requirements documents often exist in the form of user stories, which may be natural languages that do not meet standards such as SBVR.

Moreover, requirements also exist in online user reviews. Users post comments in more informal natural languages that hardly conform to any of those standards. Researchers realize the importance of user involvement in requirement elicitation and its meaning to project success a long time ago [72]. Kujala *et al.* emphasize that customers are often the most important stakeholders [72]. Recently, researchers have begun to utilize social network sites to support requirement elicitation [73]. Genc-Nayebi and Abran think users and developers have different viewpoints when using user reviews of app stores [21] - users are interested in opinions and experiences of others, whereas developers value not only those, but also missing requirements and requested app features.

Users express their requirements in the forms of new feature requests and their opinions to the software. Analysis of their opinions normally require opinion mining in current literature, which is reported in the previous section 2.1. This section will focus on new feature requests for requirement elicitation.

Researchers have used syntactic patterns, such as chunking, chinking and regular expressions to mine useful information from last century. According to Harrod [74], chunking was invented in a 1956 paper by George A. Miller. It is reported by Levithan [75] that regular expressions were invented in the 1950s by an American mathematician Stephen Kleene; In 1986, Canadian programmer Henry Spencer first released a regex library that could be freely included in other programs. The name “regex” is short for regular expressions.

The techniques that the component “Request Elicitor” of this prototype use are mainly chunking and regular expressions. Regular expressions are widely used in computer science to define search patterns. Chunking and chinking are widely used in computer science when dealing with language patterns. For example, Yi *et al.* use a set of POS patterns (chunking) to detect candidate feature terms in their work [45]. For requirement elicitation, Bazhenov is potentially the first who claims automated requirement elicitation with NLP approach chunking [76]. At the same time, Huertas and Juárez-Ramírez also built their tool in an early research for automatic requirements evaluation that uses NLP techniques such as chunking and regular expressions [77]. Iacob and Harrison proposed their approach to extract user requests from user reviews through linguistic rules in 2013 [78]. Table 9 lists some papers that use NLP techniques in requirement elicitation.

Table 9 Papers using natural language processing in requirement elicitation

Year	Paper	Methods
2005	Mining Aspects in Requirements [79]	The proposed approach can identify candidate concerns and viewpoints from informal document or requirement specification through POS tags and frequencies of occurrences.

2005	Automatic transition of natural language software requirements specification into formal presentation [80]	From text specification or text-interview, sentences are decomposed into subject, predicate, and objects, though the details on how to achieve these elements are not discussed. A tabular form of the resulting information is used and further built a class model.
------	--	---

2010	Experiments in Automated Identification of Ambiguous Natural-Language Requirements [81]	WEKA is used as the tool to distinguish the ambiguous and unambiguous requirements in the experiments that was carried out on about 500 requirements training data. Pre-processing such as POS tagging and removal of stop words were applied on the training data. The experiments then take place in WEKA using its 29 classifiers.
------	---	---

2010	Text2Test: Automated inspection of natural language use cases [82]	A prototype IDE built on Eclipse is developed to support use case authoring from natural language description. Abstract models of use cases are built in the processes from the use case text to support automated analysis and use case refinement. Context model is used in the analysis. Necessary NLP techniques to decompose the use case text are introduced in another paper.
------	--	--

2010	Automatic detection of nocuous coordination ambiguities in natural language requirements [83]	For requirement documents, a machine learning algorithm is used to determine whether an ambiguous sentence is nocuous or innocuous via syntactic patterns. The training data are based on a set of heuristics that draw on human judgments.
------	---	---

2010	Combining probabilistic tagging with rule-based	A number of NLP techniques along with chunking were used in the work. Chunking is utilized to
------	---	---

	multilevel chunking for requirements elicitation [76]	produce candidates for concept nodes in a later ontology layer, which represents requirements.
2011	NL-based software requirements elicitation and specification [84]	automated English texts are transformed to Semantic Business Vocabulary and Rules (SBVR) presentations through a pipelined architecture, a prototype tool as an Eclipse plugin.
2012	NLARE, a natural language processing tool for automatic requirements evaluation [77]	An automatic requirements evaluation architecture is proposed, which includes atomicity, ambiguity, and completeness checking. The chosen NLP techniques include at least tokenizers, spellchecker, POS tagger, chunking, and regular expressions.
2013	Is knowledge power? The role of knowledge in automated requirements elicitation [85]	The NLP and information retrieval (IR) techniques are used. NLP techniques include tokenization, POS tagging, stop word elimination and word lemmatizing. IR techniques such as vector space models are used to assign requirement categories to the extracted text bricks.
2013	Retrieving and analyzing mobile apps feature requests from online reviews [78]	Iacob and Harrison proposed an approach to extract user requests from user reviews through linguistic rules, which inspired the “request elicitor” component in this thesis.
2015	Automatic generation of UML sequence diagrams from user stories in Scrum process [86]	From user stories that contain sentences conforming to strict patterns, a simple approach is able to convert a sentence into a single message sequence diagram. The technique to recognize the patterns of the sentences is perhaps a string test.

The diagram is presented in Eclipse with the help of a UML 2 tool SDK plugin.

2016 An efficient automated design to generate UML diagram from Natural Language Specifications [87] From scenario description that contains sentences conforming to fixed patterns, algorithms are able to produce simple activity diagrams and sequence diagrams, with the help of POS and Stanford parsers.

A number of other researchers have worked on the mobile app store user reviews for opinion mining and requirement elicitation [88]–[101]. According to Tavakoli *et al.*'s survey, only a small number of studies used feature extraction techniques for mobile app reviews [22]. In spite of the small number, researchers are creative in their research methodologies and invent a number of algorithms.

Khalid manually tagged 6390 user reviews for 20 iOS apps and found 12 types of user complaints, which include requests for additional features [100].

Oh *et al.* built a SVM classifier to identify whether a user review is informative from the developer's viewpoint, which are functional bug, functional request and non-functional request. It is an interesting finding that their initial LDA results were unrecognizable, therefore they reached a conclusion that LDA cannot directly apply to this task at the time of writing [96].

Chen *et al.* also worked towards the similar question, finding informative user reviews for developers, and they also used topic modelling [94]. Chen *et al.*'s framework results indicate effectiveness.

Guzman and Maalej claim they extract fine-grained features and assign a sentiment score to each sentence [98]. They take nouns, verbs and adjectives as candidates for features, and take their collocations as features. It may be appropriate to say that these features are more like the concept of features or topics in opinion mining, rather than the concept of features or functions in requirement elicitation. They assume that the sentiment scores of the features should be the same as the sentences' sentiment values. It is arguable whether this is always

true in the longer review sentences. They also applied topic modelling on the resulting feature sets. As the features are collocations, it is understandable that the results would be different with the topic results from single word topics.

Panichella, Sorbo and Guzman manually created a taxonomy after analysis on some user reviews, developer emails, and previous researchers' work [97]. The resulting taxonomy is practical, which includes four categories: information giving, information seeking, feature request and problem discovery. They then experimented three different methods to help the classification, and concluded that their combinations reached appreciable performance.

Tian *et al.* analysed the metadata of 1492 mobile apps [90]. They input 28 factors into a random-forest classifier for high-rated app identification. The top 3 most influential factors are the install size, the number of promotional images and the target SDK version. These are reasonable and acceptable conclusions.

Different with [97], Yang and Liang work on classification of functional and non-functional requirements [99]. They combine NLP and IR techniques. The results are relatively stable at a certain size of samples.

Maalej and Nabil also work on review classification, which classifies reviews into four categories: bug reports, feature requests, user experiences and ratings [95]. Different classifiers and different combinations were tested. The results showed that combinations were better than a single classifier alone.

Gu and Kim's work [93] is the closest paper to this thesis especially for their aspect-opinion extraction. They mainly did their work with machine learning, whereas this thesis is done without machine learning approaches. They first classify user reviews into five categories: aspect evaluation, bug reports, feature requests, praise and others. The classification is achieved through a supervised machine learning approach with adoption of a Max Entropy classifier. This work is done in this thesis through a keywords array. The difference in this thesis is that the classification results can be more categories that are concept classes in the ontology. In addition, all opinions are extracted in this thesis, rather than only aspect evaluation as in Gu and Kim's work.

Gu and Kim's work also contains aspect-opinion extraction, which is similar to the "topic-opinion extractor" component in this thesis. Similar work to this usage of Stanford Parser is rare. Gu and Kim's work is the only one that has been found. Both work use almost the same tool and approach without knowing each other. In Gu and Kim's work, they first use Stanford Parser to generate a parsing tree. In their next step, a pattern-based parser is built to extract aspect-opinion pairs according to 26 predefined semantic templates. The advantages of their method are that multiple words are acceptable as both aspects and opinions; and that negation is also considered in specific patterns. Same to their work, the "topic-opinion extractor" component in this thesis also traverses the dependency tree. The patterns that this thesis uses are only direct relationships between single topic words and single opinion words, therefore the number of patterns are much more than 26 and thereby it is arguable whether they have broadened coverage. The disadvantage of this component is currently awkward in handling multiple words. The advantage of this component is that negation words are treated separately from the patterns, therefore are able to be spotted on more occasions.

Their sentiment analysis is achieved through sentiment analysis tool "Deeply Moving". Whereas this thesis borrows SO-CAL from a Canadian computational linguist Prof. Taboada [64], which perhaps is more complicated in algorithm therefore produces better calculation. Their visualization function is also good in presentation. This thesis places visualization functions after the ontology queries, therefore hands over to the system's users.

Vu *et al.* extract opinions that match templates [101]. They built the templates with POS patterns. The resulting opinion phrases are clustered later based on similarity calculations of a clustering algorithm.

Keertipati, Savarimuthu and Licorish solve a task of prioritization on mobile app features [89]. Prioritization is done based on four attributes and three approaches are explored.

McIlroy *et al.* solve a task of labelling user reviews [88]. If a user review has more than one aspect, they are able to label the reviews with more than one labels. Multiple approaches and multiple labels are combined and experimented.

2.3 Concept mapping onto ontologies

Ontology is the philosophical study of being. Although the terminology of ontology has been borrowed by computer science and information science, researchers commonly tend to avoid providing a formal definition for it in teaching. Horridge *et al.* state that “an ontology describes the concepts in the domain and also the relationships that hold between those concepts” [102].

Ontology mapping finds interrelationships or correspondences between entities, with an emphasis that is across multiple ontologies [103]. Ontology mapping can be classified into the three categories: 1) mapping between an integrated global ontology and local ontologies, 2) mapping between local ontologies, and 3) mapping on ontology merging and alignment [104].

Gilson *et al.* provide a typical good work for ontology mapping [105]. They use at least three ontologies in a pipeline structure: Domain Ontology, Semantic Bridging Ontology, and Visual Representation Ontology. The source web pages are mapped onto the domain ontology first. Then the semantic bridging ontology pipes the data in the domain ontology into the visual representation ontologies through the bridging relationships defined in it. Lastly, each visual representation ontology can produce one type of visualization based on the data definition.

Because ontology mapping is a concept for ontologies mapping to ontologies, which is different with the involved task in this thesis, this section will only introduce a typical technique that is relevant to this framework. WordNet is widely used in ontology mapping for measuring similarities among synonyms [106]. In this thesis, when matching the topic-opinion pairs from the database to the ontology “Subject” concepts, the method in this prototype is close to the matching category using a linguistic resource, WordNet, according to this survey [104].

“WordNet® is a large lexical database of English” [68]. According to Hirst and Budanitsky, many researchers have explicitly used WordNet in their Lexical Semantic Relatedness research [107]. Yet no document offers clear statistics on the huge number of software projects using WordNet.

This framework elicits user requests and extracts topics and matched opinions from user reviews using linguistic resources based on natural language processing techniques, then finally maps them onto the corresponding ontology concepts. Because researchers think the term “ontology mapping” should be used specifically for across ontologies, the process in this framework is phrased as concept mapping onto ontologies.

3 Methods

This chapter details the methodology of the proposed framework by describing the prototype. It is accomplished component by component by introductions on the designs, implementations, issues encountered and issues that need attention in future uses. These details are explained in specific chapters and sections later in this thesis.

Firstly, section 3.1 briefly describes the architecture of the system.

Section 3.2 describes the process on how the first component, a set of web sourcing code, is accomplished. This introduction is detailed to a degree that readers can write their own web downloading code after reading this section.

Section 3.3 introduces the borrowed SO-CAL component and the adjustments to adapt it into this framework. The steps to run SO-CAL in this prototype are also given in detail.

Section 3.4 depicts how the component Request Elicitor is constructed and how to maintain in the future. The questions of how to balance between the number of linguistic rules and the coverages of the rules, and how to tackle the data sparsity problem are answered in the process.

Section 3.5 reports the Topic-Opinion Extractor component in detail. This component relies on the parsing of dependency trees. Starting from an example on how it works for a dependency tree, follows the designs of how to extract the topic and opinion pairs via linguistic rules, this section details the two versions of traversal algorithm, and how the pairs are filtered for the ontology. Some interesting issues are also handled and reported, such as the integration way with the previous component “Request Elicitor”, and the mapping onto the “app” category from implied terms.

The last section 3.6 reveals the general mobile app user review ontology, an algorithm of how to populate a complex ontology from the database, and how to tackle some issues in practice.

3.1 A brief description of the architecture of the system

The keys to this framework's ability to analyse user reviews are two sets of linguistic rules, which are used for two components: request elicitor and topic-opinion extractor. Their design and rule making process are introduced in section 3.4 and section 3.5 respectively. SO-CAL, as a component parallel to these two components, calculates a sentiment value for each user review. A set of web sourcing code is used as the starting point of this framework for the source of user review data. An ontology populator is used to populate the extracted and processed data into the general mobile app user review ontology in order to facilitate further visualization, query and analysis by the users.

The overall structure after integration is a Batch Sequential style Data Flow architecture. This is shown in the figure below (Figure 1 with more detail).

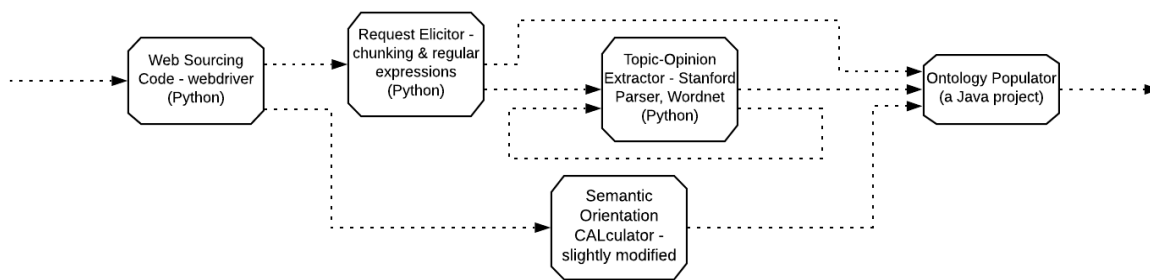


Figure 5 The framework architecture with further more detail

The five components are Web Sourcing Code, SO-CAL, Request Elicitor, Topic-Opinion Extractor, and Ontology Populator. The connectors are illustrated by the data flows above and also briefly explained below.

The input for Web Sourcing Code is the data flow containing user reviews of different mobile apps displayed on Google Play. The output of Web Sourcing Code contains downloaded user reviews, which is also inputs for Request Elicitor and SO-CAL. Data flow containing user requests is one of the outputs of Request Elicitor, and one of the inputs for Ontology Populator. Data flow of user reviews that do not contain user requests is one of the outputs of Request Elicitor, and the input for Topic-Opinion Extractor. This connector is especially explained in the next subsection 3.5.7. The Topic-Opinion Extractor has a loop that loops

through each dependency tree until the sentence is cleared for topic-opinion pairs. Data flow containing topic and opinion pairs is the output of Topic-Opinion Extractor, and one of the inputs of Ontology Populator. Data flow containing sentiment scores of reviews is the output of SO-CAL, and one of the inputs for Ontology Populator. Data flow containing populated ontology is the output of Ontology Populator.

The current framework structure avoids the problems of a packed software that will have to manage memories through settings, which potentially decreases the performance or even crashes on large amounts of data.

The main NLP tools used in this framework are NLTK (The Natural Language Toolkit) [66], SO-CAL (Semantic Orientation CALculator) [59], Stanford Parser [47], and WordNet [68].

3.2 Web sourcing user reviews

This section introduces how the set of web sourcing code is developed and how to maintain it. First of all, the author wishes to claim the legality of this component and the activities that this component is used for in this PhD program. This has been reported in section 1.4.

In this component, the user review data are copied from Google Play. During the web sourcing code's development, it is found that Google Play changes its website designs quite often, at least every few months. As a consequence, the code has to evolve each time to adapt to the new versions of the website.

The changes most frequently happen in the CSS class names or attribute names in the tags. But it is observed that Google Play made a major change in the full review page layout in 2018, possibly in March or April. Before this change, user reviews were displayed page by page by clicking a button on the full review page of an app. After this change, user reviews have been displayed on one page only. When a user reaches the end of the page, more user reviews are dynamically displayed further below the previous reviews.

This change meant that the main technique for browsing more reviews also changed, from clicking the next page button on the previous version, to scrolling down the scrollbar of the browser in the new version.

3.2.1 Web sourcing process

If researchers who use this framework aim to mine a number of mobile apps according to certain criteria, and do not know which mobiles apps they are before the web sourcing, they will typically go through the following steps:

- (1) Select a category of mobile apps or search by keywords
- (2) Run the code C.2 to source the metadata of each app on the above search result page
- (3) Apply the inclusion and exclusion criteria onto the resulting spreadsheet
- (4) For the selected apps, run the code C.3 to source the metadata on the app pages
- (5) Run the full review page sourcing code C.4 on the included mobile apps

Taking an example of the third dataset used in the evaluation, the above process can be illustrated in more detail below.

Firstly, the author goes to the Google Play app store and searches mobile apps by the key string “driving theory”. The results page contains a number of mobile apps for driving theory learning and testing. At the time of writing, the maximum number of apps that could possibly return on one search results page is 250.

Secondly, the code to source the metadata of each app on the above search result page, in the code listing C.2, needs to run. This code takes the URL of the above search results page as the input, and outputs the metadata into two files, one text file and one csv file with identical content.

Importantly, before running code C.2, the users of this system have to make sure this line of code points to the correct installation of FireFox version 47. The installation guide for FireFox 47 is in Appendix A 8.1.

Code Snippet 1 Code pointing to FireFox 47

```
binary = FirefoxBinary('C:\Program Files (x86)\Mozilla Firefox\Firefox.exe')
```

In order to successfully run the web sourcing code, the geckodriver version is also fixed to v0.11.1. FireFox browser and the matched geckodriver are the only two software that have fixed version requirements at the time of writing. All the necessary software’s installation guides are in Appendix A Software installation.

Code C.2 uses a dummy loop to fully scroll the whole page down to the end first, then uses another loop to enter each mobile app page from the search results. Controlling the scrollbar is achieved by a line of JavaScript scrolling the page downwards by a number of pixels combined with a timeout of 5000 milliseconds. The code snippet is listed below, and is wrapped into the dummy loop:

Code Snippet 2 Code scrolling the web page downwards

```
brw.execute_script("window.scrollTo(0, 15000000);")  
timeout = 5000
```

The copying process for metadata from each of the app pages is wrapped into the second loop. The code finds all the links of app results by XPath values, then loops through each of

the links, extracts the metadata by XPath on each app page, and finally outputs these metadata into a spreadsheet, along with an output text file. The method of locating elements and extract values by XPath is explained in the next subsection 3.2.2 “Mobile app page metadata sourcing”. The code snippet below describes the process of producing a csv spreadsheet with the data.

Code Snippet 3 Code producing a csv spreadsheet with the data

```
with open('GooglePlayReviews/GooglePlayDrivingTheoryAppsSelection.csv', 'w',
newline='') as csvfile:

    fieldnames = ('App Name', ....., 'Link')

    writer = csv.DictWriter(csvfile, fieldnames=fieldnames)

    writer.writeheader()

    l=0

    while l<len(urls):

        ## Omitted code within the while loop copying the metadata from each app page.

        writer.writerow({'App Name':appName, ....., 'Link':urls[l]})

        l=l+1
```

It is worth noting, “newline=’’” should be kept in the first line above, which will prevent an empty line between every two lines of records.

Thirdly, the author applies some inclusion and exclusion criteria on the resulting spreadsheet. In practice, researchers can decide their inclusion and exclusion criteria according to their research aims. As an example, the author chose the criteria below for this dataset.

If Overall Rating > 4, Number of Ratings > 5000, and Number of Installs >= 50000, then the inclusion conclusion is “Yes”.

Because the selection process is mainly accomplished in a spreadsheet that can be opened in Excel, the inclusion and exclusion criteria can be expressed in a formula below:

Table 10 an example inclusion criteria formula

Inclusion criteria = IF(C2>=4, IF(D2>5000, IF(I2>=50000, "Yes", "No"), "No"), "No")

In the above formula, column C is Overall Rating, column D is Number of Ratings, and column I is the Number of Installs.

If users of this system wish to keep this formula in the spreadsheet, and copy the filtered list of apps to another worksheet in the same file, it is necessary to change the file format from csv to a common excel file format.

In this example, 10 out of the 250 mobile apps were selected according to the above criteria. They are listed in Table 11:

Table 11 Included driving theory mobile apps from the third dataset

App Name	Overall Rating	Number of Ratings	Updated On	Number of Installs
Driving Theory Test 4 in 1 Kit + Hazard Perception	4.7	14,633	15-Aug-19	100,000+
Driving Theory Test Free 2019 for Car Drivers	4.2	9,600	31-Jul-19	1,000,000+
Driving Theory Test 2019 Free for UK Car Drivers	4.5	7,086	31-Jul-19	100,000+
Theory Test 2019	4.3	10,368	15-Apr-19	1,000,000+
Car Driving School Simulator	4.5	180,562	22-Mar-19	10,000,000+
Driving Academy - Car School Driver Simulator 2019	4.3	39,993	4-Aug-19	5,000,000+

City Car Driving & Parking School Test Simulator	4.3	68,790	4-Jul-19	5,000,000+
Fahren Lernen - Your driver's license training	4.6	81,823	11-Apr-19	1,000,000+
Driving in Car	4	35,216	15-May-19	5,000,000+
Driver Test: Crossroads	4.5	30,160	3-Jan-19	5,000,000+

The subsection 3.2.2 “Mobile app page metadata sourcing” describes the next step for the app page sourcing. The subsection 3.2.3 “Full review page sourcing” after that is the last step for the full review copying.

3.2.2 Mobile app page metadata sourcing

A typical mobile app page contains the metadata of the app. Figure 6 shows the metadata that this web sourcing code mines.

Once the sourcing target data are decided, a developer tool is needed to locate the correct tags of the data. At the time of writing this code, during 2016 to 2018, Firebug helped mining the XPath of page elements. But at the time of submission of this thesis, Firebug has been deprecated. Instead, Firefox and Chrome both have developer tools containing similar functions to Firebug. For FireFox, it is “Inspector” under “Web Developer”. For Chrome, it is “Developer Tools” under “More Tools”. The developer tools in these two major popular browsers are useful not only for finding the XPath, but also for being able to copy the XPath. They both use the same icon as highlighted in the Figure 7.

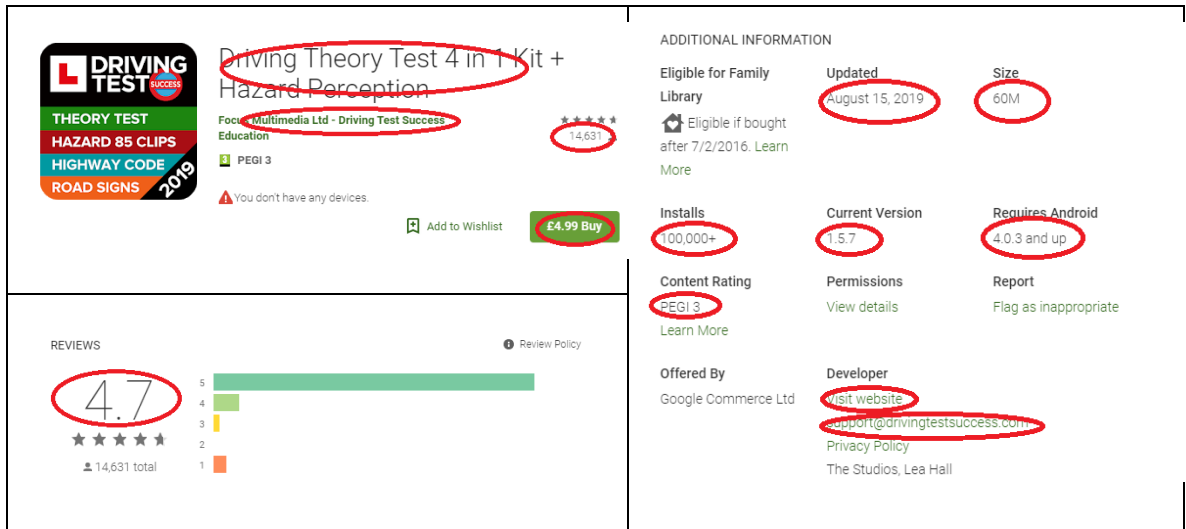


Figure 6 Metadata being mined on a typical mobile app page

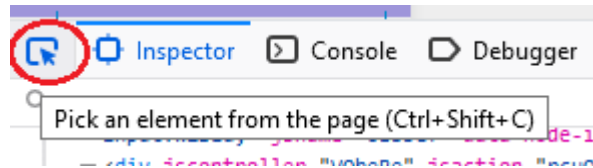


Figure 7 Icon for the tool locating webpage elements

However, the XPath values that these developer tools return in the copy functions are absolute, therefore not always useful. This is because web page structure changes almost page by page. Most of the time, the developer tools are helpful for locating the right elements, then the users of the system need to analyse the XPath and decide the relative XPath values that allow the code to find the elements across all app pages.

Having decided the relative XPath values, elements can be found by groups or individually through the following two methods: `find_elements_by_xpath()` and `find_element_by_xpath()`. The former returns all elements that match the XPath. The latter method will return the first element that matches the condition.

Code Snippet 4 Getting elements by group or individually

```
apps = brw.find_elements_by_xpath('//div[@class="Vpfgmd"]')

price = brw.find_element_by_xpath('//span[@class="oocvOe"]//button').text
```

Table 12 lists the relative XPath values of typical metadata on a mobile app page that work on 27th December 2019.

Table 12 XPath values of typical metadata on 27th December 2019

Mobile app name	//h1[@class="AHFaub"]
Developer Company	//div[@class="jdjqLd"]//span[1]
Number of Ratings	//span[@class="AYi5wd TBRnV"]//span[1]
Pricing	//span[@class="oocvOe"]//button
Average Stars	//div[@class="BHMmbe"]
Updated Date	//div[@class="JHTxhe IQ1z0d"]//*[text()='Updated']/following-sibling::span[@class="htlgb"]
Size	//div[@class="JHTxhe IQ1z0d"]//*[text()='Size']/following-sibling::span[@class="htlgb"]
Number of Installs	//div[@class="JHTxhe IQ1z0d"]//*[text()='Installs']/following-sibling::span[@class="htlgb"]
Current Version	//div[@class="JHTxhe IQ1z0d"]//*[text()='Current Version']/following-sibling::span[@class="htlgb"]
Required Android Version	//div[@class="JHTxhe IQ1z0d"]//*[text()='Requires Android']/following-sibling::span[@class="htlgb"]

Content Rating	//div[@class="JHTxhe IQ1z0d"]//*[text()="Content Rating"]/following-sibling::span[@class="htlgb"]//span[@class="htlgb"]/div[1]
In-app Products Pricing	//div[@class="JHTxhe IQ1z0d"]//*[text()="In-app Products"]/following-sibling::span
Developer Website	//div[@class="JHTxhe IQ1z0d"]//*[text()="Developer"]/following-sibling::span[@class="htlgb"]//a[starts-with(@href, "http")]
Developer Email	//div[@class="JHTxhe IQ1z0d"]//*[text()="Developer"]/following-sibling::span[@class="htlgb"]//a[starts-with(@href, "mailto:")]

This component uses the webdriver module from Selenium. Selenium provides a number of methods to locate elements in a webpage. These methods can locate elements by id, name, xpath, link_text, partial_link_text, tag_name, class_name, and css_selector. However, only xpath and css_selector from this list are currently provided with the copy function across both developer tools from Firefox and Chrome. The css_selector is more inconsistent and difficult to locate across different web pages than XPath. The reason for this is that css_selector is mainly composed of a css element style followed by a series of position numbers of the target element in the sequences of the lists. As the css_selector example below indicates, this is very difficult to keep stable across webpages:

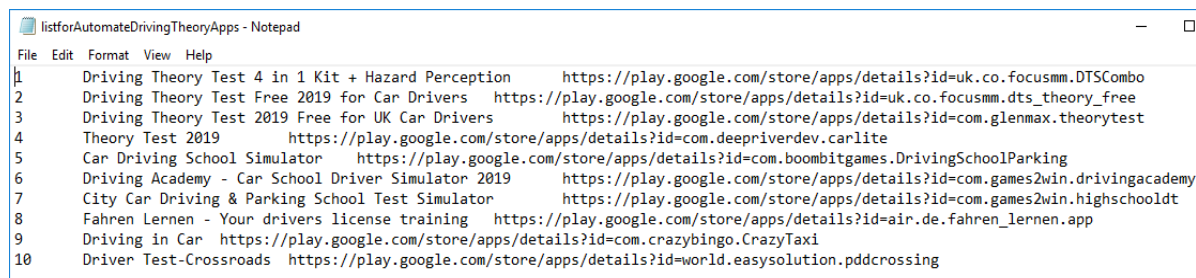
```
li.card-outer:nth-child(1) > a:nth-child(1) > div:nth-child(1) > div:nth-child(1) > div:nth-child(1)
```

If the users of this system have not created the database as defined in the first step in Appendix B Steps to run this framework, this is the latest time to do it. This is because the sourcing code for the mobile app page will output the metadata to a number of text files and into the database at the same time.

The input of this piece of code C.3 is a list of apps that have passed the inclusion criteria from the above step. The code goes through each of the apps in turn and extracts the same

metadata fields. The outputs of this code are a number of text files that are named with the app names, and the populated “apps” table in the database.

The input file of the app list has three pieces of information, serial number “ID”, “app name”, and the “link” to the app page. Each line represents one app. The simplest way to produce this input file is to copy the list of mobile apps that have passed the inclusion criteria from the spreadsheet in the previous step, and only keep the information of the above three columns. The figure below shows an example of the resulting input text file that the author used in the third dataset.



```
listforAutomateDrivingTheoryApps - Notepad
File Edit Format View Help
1 Driving Theory Test 4 in 1 Kit + Hazard Perception https://play.google.com/store/apps/details?id=uk.co.focusmm.DTSCombo
2 Driving Theory Test Free 2019 for Car Drivers https://play.google.com/store/apps/details?id=uk.co.focusmm.dts_theory_free
3 Driving Theory Test 2019 Free for UK Car Drivers https://play.google.com/store/apps/details?id=com.glenmax.theorytest
4 Theory Test 2019 https://play.google.com/store/apps/details?id=com.deepriverdev.carlite
5 Car Driving School Simulator https://play.google.com/store/apps/details?id=com.boombitgames.DrivingSchoolParking
6 Driving Academy - Car School Driver Simulator 2019 https://play.google.com/store/apps/details?id=com.games2win.drivingacademy
7 City Car Driving & Parking School Test Simulator https://play.google.com/store/apps/details?id=com.games2win.highschooltd
8 Fahren Lernen - Your drivers license training https://play.google.com/store/apps/details?id=air.de.fahren_lernen.app
9 Driving in Car https://play.google.com/store/apps/details?id=com.crazybingo.CrazyTaxi
10 Driver Test-Crossroads https://play.google.com/store/apps/details?id=world.easysolution.pddcrossing
```

Figure 8 An example of the input file containing a list of apps

Although Python has implicit type conversion, which automatically recognizes and converts some data types, such as numbers and strings, this does not apply to “date” type. The code in Code Snippet 5 shows an example of getting a metadatum “updated date” string by XPath and converting it into the “date” type for MySQL.

The XPath that the browser arrives at in the example below through the `find_element_by_xpath` method is a relevant XPath value. Firstly, it looks for a div element with a class name “X”. Secondly it looks for any child element of the previously found div element regardless of the type if the child element has a text value “Updated”. Thirdly, in this child element’s following siblings, it looks for a span with a class name “Y”. Eventually, the text value of this span is the target value, which is the “updated date” from the app page. Because the string of the updated date contains a comma, it has to be removed, which is achieved by the `“.replace(",", "")”`.

The next line of code converts the string value into a date value, because the “updatedOnDate” in the “apps” table is defined as the MySQL type “date”. The second

parameter of the “strptime” method is the format of the source date, namely, to interpret the first parameter as the format of “month date year”.

It is worth mentioning that '%B' matches a full month name, such as December. '%b' matches an abbreviation of a month, for example Dec. '%Y' matches a four-digit year, whereas '%y' matches a two-digit year. The second parameter defines exactly how the first parameter should be interpreted in format, including spaces and punctuation. As the first parameter that the upper line of code produces is in the format of “August 15 2019”, the second parameter has to be '%B %d %Y'. For example, if the upper line of code does not remove the comma from “August 15, 2019”, the second parameter for this must be '%B %d, %Y'.

Code Snippet 5 Getting “update date” by XPath and converting to “date” type for MySQL

```
from datetime import datetime
## Omitted code not a part of this snippet, see the list of code C.3
updatedDate = \
brw.find_element_by_xpath('//div[@class="X"]//*[text()="Updated"]/following-
sibling::span[@class="Y"]').text.replace(",","")
updatedOnDate = datetime.strptime(updatedDate, '%B %d %Y')
```

Code Snippet 6 Outputting metadata to a text file

```
## Omitted code not a part of this snippet, see the list of code C.3
outputFilename = 'GooglePlayDrivingTheoryReviews/00'+appId+'_'+data[1]+' .txt';
url = data[2];
brw.get(url)
outputFile = open(outputFilename,'w')
outputFile.write('\n'+url+'\n'+'\n')
appName = brw.find_element_by_xpath('//h1[@class="AHFaub"]').text
outputFile.write("App Name: " + '\n')
outputFile.write(appName + '\n' + '\n')
```

```
## Omitted code not a part of this snippet, see the list of code C.3
outputFile.close()
```

The snippet of code in

Code Snippet 6 describes the process of writing the metadata, such as `appName`, into a series of text files with the serial number of the app and the app name as the output file names.

The code in Code Snippet 7 inserts data from Python 3 into a MySQL database. The “`add_app`” assignment defines the SQL statement. The “`data_app`” assignment defines the mappings between the metadata and the fields in the “`apps`” table in a dictionary format. The keys of the dictionary are field names, and the values are extracted metadata.

Code Snippet 7 Outputting metadata to the database

```
myConnection = \
pymysql.connect(host=hostname,user=username,password=password,db=database)
cursor = myConnection.cursor()
## Omitted code not a part of this snippet, see the list of code C.3
add_app = ("INSERT INTO apps (appName, ....., link)"\
          "VALUES (%(appName)s, ....., %(link)s)" )
data_app = {'appName':appName, ....., 'link':url}
cursor.execute(add_app, data_app)
myConnection.commit()
```

3.2.3 Sourcing the full review pages

The last step of web sourcing is to download the reviews from the full review page of each app. On the current app page’s review area, there is a “`READ ALL REVIEWS`” link below the area. Clicking this link will open the full review page, and the URL of the app page will appear with a suffix “`&showAllReviews=true`”. Therefore, the input file of the app list used in the previous step is still valid in this step. The only change needed is to concatenate the URLs with the suffix.

The figure below shows the typical review data that are mined as of 27th December 2019.

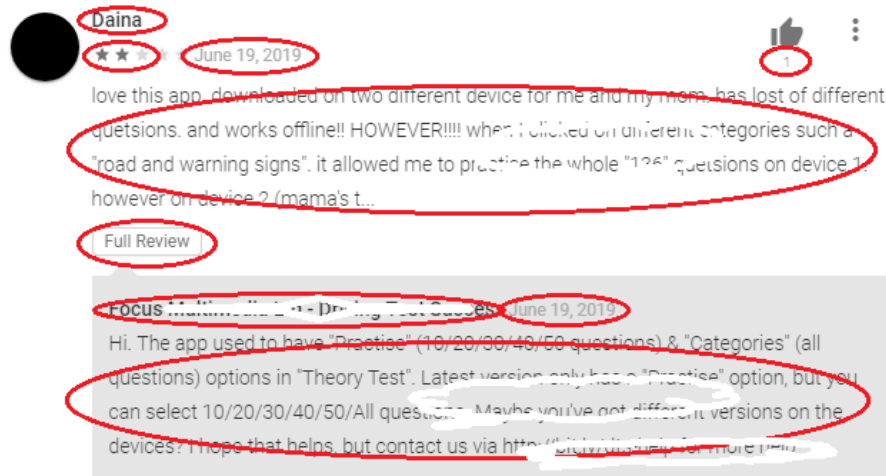


Figure 9 Review data being mined on a typical review

If the text is longer than the length limit set by Google Play, the reviews are not fully displayed initially on the full review page. In this case, a “Full Review” button shows. Clicking on this button will display the complete text of this review. The web sourcing code of this page is doing the same thing as a user would do when browsing the reviews manually. It looks for a “Full Review” button for each review, and clicks it if there is one. Then it copies the review text from the XPath whichever available. The table below lists the typical review data and their relative XPath values that work as of 27th December 2019.

Table 13 XPath values of typical review data on 27th December 2019

review_author_name	./span[@class="X43Kjb"]
review_rating	(./div[@role="img"]).get_attribute("aria-label")[6]
review_date	./span[@class="p2TkOb"]
number_of_helpful	./div[@class="jUL89d y92BAb"]
review_title	./span[@class="IEFhEe"]
Full review button	./button[@class="LkJZd ScJHi OzU4dc "]

Review text if there is no “Full Review” button	<code>./span[@jsname="bN97Pc"]</code>
Review text if there is a “Full Review” button	<code>./span[@jsname="fbQN7e"]</code>
Replied developer name	<code>./span[@class="X43Kjb"]</code>
Developer replied date	<code>./span[@class="p2TkOb"]</code>
Developer replied text	<code>(./div[@class="LVQB0b"]').text.split(replyDate, 2)[1]</code>

3.2.4 Controlling the scrollbar

How to control the scrollbar is a question initially encountered on the search result page, where the apps will only appear fully when the scrollbar is scrolled until the end of the page. However, on the full review page, there are many more reviews, and sometimes a “SHOW MORE” button appears when the scrollbar is scrolled multiple times. Therefore, multiple loops are used on this page to control the page scrolling.

Similar to the previous app search result page, a dummy loop is used to scroll the page down as much as possible at the beginning. The same line of JavaScript to scroll the scrollbar downwards is wrapped in this loop as well.

Next, a try-except block statement is used to handle the main logic of the code. The “try” block deals with the “SHOW MORE” buttons at the same time when scrolling the scrollbar sufficiently in between two “SHOW MORE” buttons. The code looks for such a button first. Once it appears, a single line of JavaScript is used to scroll the scrollbar until this button appears into the view, as shown in Table 14.

Table 14 JavaScript controlling scrollbar to an element

```
brw.execute_script("arguments[0].scrollIntoView(true);", element)
```

Then the button is clicked. From this moment until this button appears again, another dynamic loop is used to scroll the scrollbar downwards. Because the response time here is longer and longer, and more and more reviews appear, this loop has to increase the number of times of the JavaScript that scrolls down and the timeout being executed. A weight number is multiplied by an increasing count number, which counts the number of times that the “SHOW MORE” buttons are clicked. During the author’s web sourcing process for the “Driving Theory” dataset, the weight is set as 90. This sufficiently allows the full review pages to expand until the end of all reviews on the author’s computer.

Different from the purpose for which the “except” block is designed, this code puts the main reviews downloading code in the “except” block. This will ensure that all reviews will only be downloaded when no more reviews are possible, when the condition of the “except” block is triggered by either `StaleElementReferenceException` or `NoSuchElementException`.

Inside the main code of extracting reviews, the code extracts the data for a review and outputs them into a text file and the database “reviews” table first, then uses another try-except block to extract the developer reply if there is one for this review.

Importantly, if a review has a developer reply associated, when inserting the developer reply data into the database “developerreply” table, it has to have the “id” of the current review. This is compulsory because “reviewId” is a foreign key in the “developerreply” table with the reference to the “id” of the “reviews” table. However, the field “id” in the “reviews” table is an auto_increment value that the Python code does not know when inserting the reviews into the database. The line of code below is very useful to resolve this problem. The “lastrowid” method of the cursor gets the “id” value from the record that has been successfully inserted into the database and returns it. It is also important to place this method below the above line that commits the changes.

Code Snippet 8 Method getting “id” of the existing review for the associated developer reply

```
myConnection.commit()

id = cursor.lastrowid
```

When copying each review, if there is a developer reply associated, it is dealt with. Otherwise, a “pass” will bypass the “except” block without creating any problems. This is because there is no other statement on the same level that could accept any value from the “pass”, nor is there any database insertion statement here.

Algorithm 1 Pseudocode of the loops controlling the scrollbar on the full review page

Input: A link of a mobile app’s full review page

Output: All user reviews of this mobile app

weight = 90

try

find all reviews initially shows. . . ;

while this list of reviews is not empty **do**

scroll down the scrollbar 150000000 pixels

end while

except ElementNotVisibleException

print ("Timed out waiting for page to load")

end try

count = 1

try

look for a ‘Show More’ button;

while there is a such button **do**

scroll down the scrollbar to this button

click this button

count = count +1

s = 0;

while s < weight * count **do**

scroll down the scrollbar 150000000 pixels

timeout = 5000 milliseconds


```
s = s + 1
end while
end while
except StaleElementReferenceException, NoSuchElementException
    main code to download all reviews
end try
```

One more thing that needs to be mentioned: it seems that the “except” block of the review downloading branch comes earlier when other programs are using considerable memory on the same computer. It is not clear how the browser is affected. Therefore, users of this system are recommended to adjust the weight number of the loop in experiments on their own computer, and are advised to leave the web sourcing code to run with no other programs running at the same time.

3.2.5 Dealing with emoji

Users sometimes use emoji in their reviews. However, the default encoding of both a text file and a MySQL database is UTF-8, which does not accept emoji. Moreover, it might challenge the later NLP parsers for performance if encoding them into specific characters at the beginning. The author hopes other researchers could come up with better plans that enhance the user review analysis in this framework with emoji analysis.

In this framework’s first implementation, a simple way that ignores emoji is adopted. However, this only happens for the third dataset, “Driving Theory” mobile apps. In the first two datasets, emoji is initially replaced with a question mark “?”. Because users frequently use several consecutive emoji, the resulting question marks are also consecutive. When the Stanford Parser encounters consecutive question marks, the dependency trees are normally wrong as a consequence. This affected the parsing results of the Stanford Parser by approximately 2%. The precision of POS tagging is affected much less than that by the author’s estimation.

Therefore, the current method dealing with emoji in this framework is to ignore them. The Table 15 lists the coding methods to ignore emoji and to replace with “?” respectively.

Table 15 Methods to ignore or to replace emoji with "?"

To ignore emoji	<code>text = text.encode('ascii', 'ignore')</code>
To replace emoji with "?"	<code>text = text.encode('ascii', 'replace')</code>

The resulting texts are acceptable to the MySQL database. However, if they are output into a text file, they still need to be converted as strings. An example is below:

```
outputFile.write(str(text))
```

3.2.6 Methods dealing with issues appearing during the code running

(1) Running on an ordinary computer without massive memory

It is a challenging task for an ordinary computer's memory to run the web sourcing code. Although the automatic garbage collection has been enabled in the Python code by the statement `"gc.enable()"`, the browser still accrues large memory usage. It is observed that the computer slows down when more reviews are being downloaded along the list of apps. In order to overcome this problem, the list of apps can be shortened, even to only one app item each time. Then restarting the computer between code executions could help significantly. Leaving the computer to run the web sourcing code only, without any other programs using memory, would also be a great help.

(2) Being suspected as a robot

This is an issue that could occur at any time with regard to the web sourcing code. When it happens, either the browser freezes for a long time, or the IDLE crashes with an error message mentioning `"marionette"`. During the process of the `"Driving Theory"` dataset reviews downloading, the author did not find that Google Play aggressively detected robots and blocked the code. However, this did occur previously during the metadata mining for apps on the search result page. Once this occurs, it is helpful to swap the browser creation statement to another statement that contains a setting to set robot as `False`. In the code that the author use, this is:

```
brw = webdriver.Firefox(firefox_binary=binary, capabilities={"marionette":False})
```

(a)

Once this piece of code starts running, it is safe to stop it and swap back to the original code:

```
brw = webdriver.Firefox(firefox_binary=binary)
```

(b)

It is not found to be helpful if always keeping the code (a). In other words, it will still be suspected as a robot at some point later, and it could slow down the code significantly. Instead, swapping between the two pieces of code actually helps with this issue.

(3) Unresponsive script issue before the exception block is triggered

If the “exception clause” message is not shown on the Python IDLE, the exception block is not triggered. Before the exception block is triggered, the main part of the review downloading code does not run. At this point, the code is simply trying to scroll the webpage down as much as possible in order to get to the end of the reviews. No reviews of this app are written into the database. Therefore, it is safe to stop the unresponsive script and close the browser at this time.

In terms of the list of apps, any apps that have previously been processed successfully should be removed. This will leave the current app at the top of the list.

A reboot of the computer is strongly recommended here. Then the code can run with the new list of apps.

(4) Unresponsive script issue after the exception block is triggered

If the “exception clause” message is displayed, the exception block has been triggered. The size of the text file being named with the current app under the target folder starts to grow. At this time if an unresponsive script message is observed on the browser, the size of the text file is highly likely to be frozen at a size larger than 1 KB.

When the exception block is triggered, the code has started writing reviews and the associated developer replies to the database before it freezes. Because it is difficult to tell which reviews and developer replies of this app will be duplicated in the second running of the code, it is

better to remove the existing review and developer reply records of this app from the database and allow the code to do a fresh run. However, the “id”s of the reviews and developer replies are auto_increment, therefore there will be gaps in the “id” values after removing the unwanted reviews and developer replies. In order to overcome this issue, the auto_increment counters have to be reset to values immediately following the last review and the last developer reply’s “id”s of the previous app.

For example, the code freezes in the exception block when downloading reviews from app 8, then the users of this system want to redo the downloading of reviews from app 8 without leaving a gap in the ids across all tables. Firstly, the browser, the geckodriver window and the IDLE of the Python code need to be terminated. Then the process that follows can be illustrated with the SQL queries below:

Table 16 SQL queries dealing with unresponsive script issue if the exception block is triggered

SQL query	Result
select min(id) from reviews where appld = 8;	57658
delete from developerreply where reviewId>=57658;	Query OK, 224 rows affected (0.01 sec)
select max(id) from developerreply;	13718
alter table developerreply auto_increment = 13719;	Query OK, 0 rows affected (0.01 sec)
delete from reviews where id>=57658;	Query OK, 2565 rows affected (0.03 sec)
select max(id) from reviews;	57657
alter table reviews auto_increment = 57658;	Query OK, 0 rows affected (0.01 sec)

It is worth noting that the auto increment value can be reset to a value that is less than or equal to the maximum value currently in the AUTO_INCREMENT column, then the actual value could be reset to the current maximum plus one. However, whether this works varies for different databases. It is necessary to check the documentation of the database in use before running the statement if the value of resetting auto increment to is decided to be less than or equal to the current maximum value. Alternatively, it is always safe to reset the value to the current maximum value plus one, just as the statements in Table 16 do.

After the execution of the above SQL queries, it is necessary to modify the input file of the list of apps, and to restart the computer. Rerunning the code C.4 again at this time will not create any gaps in the database for the “id”s.

3.3 SO-CAL and its slight adjustments in this framework

SO-CAL [67] is a sentiment analysis tool developed by a research group chaired by Professor Maite Taboada at Simon Fraser University, Canada. SO-CAL is a lexicon-based tool using Stanford CoreNLP [108] and a number of dictionaries. Its source code is publicly available on GitHub [64]. Professor Maite Taboada gave the author of this framework helpful advice and permission to use SO-CAL in this research.

SO-CAL works as a component in this framework to calculate sentiment scores for reviews. Because SO-CAL can split a text into sentences and calculate a sentiment score for each sentence, the author also uses this function to split each review into sentences and calculate sentiment scores for them. Although the sentiment scores for sentences are not populated into the ontology in this first implementation, they are already in the code and database for other researchers if they need.

Because SO-CAL uses dictionaries, improving the dictionaries or adding domain-related dictionaries could improve performance. This can be a further task if researchers aim to adapt SO-CAL into specific domains.

The original source code of SO-CAL is provided for Linux or similar OS users. Therefore, the author of this framework created a batch file for Windows. Slight adjustments are also made to allow SO-CAL to write sentences, sentiment scores for both reviews and sentences into the database.

For the installation of SO-CAL, readers are referred to its GitHub page [64]. The next subsections introduce the slight adjustments that have been made to SO-CAL, and the steps to run SO-CAL specifically in this framework.

3.3.1 Creating Windows Batch files

If SO-CAL is used in Windows, two Shell scripts have to be converted into Windows Batch files, `run_text_preprocessing` and `run_sentiment_calculator`. Only two points below need to be performed for these two files for this purpose.

(1) Replacing the text “python3.5” with “py -3”;

(Depending on the Python versions on the current computer, this parameter varies. If there is only the latest version of Python 3 installed, the above parameter can be “python”. This also applies to point 3 in the next subsection 3.3.2.)

It is also important that the system environment variables recognize the Python version that is used.)

(2) Giving the new files an extension name as “.bat”.

3.3.2 A few fixes of SO-CAL in the new environment

(1) For SO-CAL-master\Source_Code\text_preprocessing\preprocess.py

Inside class Preprocess() def __init__(self, args):

Change the code between line 19 and 28 to the code as shown in Code Snippet 9.

Code Snippet 9 Fix for preprocess.py

```
# Initialize input, output_folder, standford_annotators, log_path (optional)
self.input = os.path.abspath(args.input_path)
print(self.input.strip(""))
self.input = self.input.strip("") #Juan Wang's code
self.output_folder = os.path.abspath(args.output_path)
self.output_folder = self.output_folder.strip("") #Juan Wang's code
self.input_type = "null" #Juan Wang's code
if (not (os.path.exists(self.output_folder))):
    os.makedirs(self.output_folder)
self.log_path = os.path.abspath(args.log_path)
self.standford_annotators = args.annotators
# print(self.input)
if (os.path.isdir(self.input) == True):
    self.input_type = "dir"
```

```
elif (os.path.isfile(self.input) == True):  
    self.input_type = "file"
```

Above code will fix the `AttributeError` of 'Preprocess' that object has no attribute 'input_type', and the error not being able to identify input file or folder due to a redundant apostrophe suffix.

(2) For `\SO-CAL-master\Source_Code\run_text_preprocessing.bat`

Please remove the `stanford_annotators` input data format:

```
-a 'tokenize,ssplit,pos'
```

This will prevent complaints for data input's unmatched format or empty format. Without this setting, the `preprocess.py` code will use the default setting, which never causes errors during the author's experiments.

(3) For `SO-CAL-master\Source_Code\sentiment_calculator\SO_Run.py`

Change line 116 from

```
with open(file_sentiment_path, 'a') as csv_out:
```

to

```
with open(file_sentiment_path, 'a', newline='') as csv_out:
```

if the output csv file is expected as a compact layout without an empty line between each two lines of records.

Change line 205 from

```
cmd = "python3.5 sentiment_calculator/SO_Calc.py -i \"" + input_path + "\" -bo \"" +  
basicout_path + "\" -ro \"" + richout_path + "\" -c \"" + config_file + "\""
```

to

```
cmd = "py -3 sentiment_calculator/SO_Calc.py -i \"" + input_path + "\" -bo \"" +  
basicout_path + "\" -ro \"" + richout_path + "\" -c \"" + config_file + "\""
```

Change line 212 from


```
cmd = "python3.5 sentiment_calculator/SO_Calc.py -i \" + file_path + "\" -bo \" +  
basicout_path + "\" -ro \" + richout_path + "\" -c \" + config_file + "\"
```

to

```
cmd = "py -3 sentiment_calculator/SO_Calc.py -i \" + file_path + "\" -bo \" +  
basicout_path + "\" -ro \" + richout_path + "\" -c \" + config_file + "\"
```

Depending on the Python versions on the current computer, this parameter varies. If there is only the latest version of Python 3 installed, the above parameter can be “python”.

Change line 237 – 244 from the left code to the right code as below.

Code Snippet 10 SO_Run.py line 237- 244 change

<pre>if gold_dct[file_name] == pos_mark: p_total += 1 if predicted_sentiment == pos_mark: p_correct += 1 elif gold_dct[file_name] == neg_mark: n_total += 1 if predicted_sentiment == neg_mark: n_correct += 1</pre>	<pre>if(file_name!="tempfile2"): if(gold_dct[file_name] == pos_mark): p_total += 1 if predicted_sentiment == pos_mark: p_correct += 1 elif gold_dct[file_name] == neg_mark: n_total += 1 if predicted_sentiment == neg_mark: n_correct += 1</pre>
--	---

The reason for this change to rule out file_name “tempfile2” is that there might be an extra file “tempfile2” created in the previous preprocessing step if something goes wrong in the input. This change is optional because it does not always happen. However, if it happens, this “tempfile2” file is empty and has no extension name, which will break code SO_Run.py.

3.3.3 Adjustments to enable writing sentences and sentiment scores to the database

For SO-CAL-master\Source_Code\sentiment_calculator\SO_Calc.py

- (1) Before or after the imports, please put these lines:

```
hostname = 'hostname'
username = 'username'
password = 'password'
database = 'databasename'

import pymysql

import pymysql.cursors
```

(2) Put database connection at the beginning of “### Main script ###”

```
# 2018-01-07 Juan Wang @ Oxford Brookes University

myConnection =
pymysql.connect(host=hostname,user=username,password=password,db=database)

cursor = myConnection.cursor()

# end
```

(3) Replace the code from the original line 1780 to the end with the code below.

Code Snippet 11 Code writing sentences and sentiment scores to the database

```
# 2018-01-07 Juan Wang @ Oxford Brookes University
reviewId = os.path.basename(args.input).strip(".txt")
# end
if output_sentences:
    richout.write("-----\nSO by Sentence\n-----\n")
    for i in range(len(boundaries)):
        # 2018-01-07 Juan Wang @ Oxford Brookes University
        sentence = get_sentence(boundaries[i] -1)
        sentenceId = i+1
        richout.write(sentence + " ")
    # end
```

```

##      richout.write(get_sentence(boundaries[i] -1) + " ")
      if i in sentence_SO:
          # 2018-01-07 Juan Wang @ Oxford Brookes University
          SOscore = float(sentence_SO[i])
          if (i<2):
              keysentence = 1
          elif (abs(SOscore)>2.0):
              keysentence = 1
          else:
              keysentence = 0
          richout.write(str(SOscore) + "\n")
          print('reviewId'+str(reviewId)+ 'sentenceld'+str(sentenceld)+
'text'+str(sentence)+ 'SOscore'+str(SOscore)+ 'keysentence'+str(keysentence))
          cursor.execute("""
              insert into tokenizedsentences (reviewId, sentenceld, text, SOscore,
keysentence)
              values (%s,%s,%s,%s,%s)
              """, (reviewId,sentenceld,sentence,SOscore,keysentence)
              )
          myConnection.commit()
      ## end.

##      richout.write(str(sentence_SO[i]) + "\n")
      else:
          richout.write("0\n")

if output_calculations:
    richout.write("-----\nTotal SO: " + str(text_SO) + "\n-----\n")
    # 2018-01-07 Juan Wang @ Oxford Brookes University

```

```

SOscore_review = text_SO
cursor.execute("""UPDATE reviews
                SET SOscore=%s
                WHERE id = %s
                """, (SOscore_review, reviewId))
myConnection.commit()
# end
if adv_learning and new_adv_dict: # output the new adverb
    f = open(adv_dict_path, "a") # dictionary
    for adverb in new_adv_dict:
        f.write(adverb + "\t" + str(int(adv_dict[adverb])) + "\n")
    f.close()

basicout.close()
richout.close()
# 2018-01-07 Juan Wang @ Oxford Brookes University
cursor.close()
myConnection.close()
# end

```

3.3.4 Steps to run SO-CAL in this framework

The conditions for SO-CAL to run in this framework are that the database has been set up and user reviews have been copied from Google Play into the database. For the full steps to run this framework, please refer to Appendix B Steps to run this framework.

The steps to run SO-CAL in this framework are listed below.

- (1) Running code C.5 to convert reviews from database to text files for SO-CAL.

The purpose of this piece of code is to provide the reviews to SO-CAL in the format that it requires. It is necessary for the readers to adjust the database connection parameters (hostname, username, password, database) according to their own database details.

The code snippet below is the part of creating the required review files and folders. It is necessary to manually set `appid` as the `k` value below. The code will create a folder with this `appid` as the name. Then the reviews of this app will be taken from the database, and written into text files separately under this folder. Each text file contains one review, and the name of the file is the ID of the review. Occasionally, people write reviews into the title field rather than the text field. If this happens, the texts in the title fields will be written into the text files. This current code needs to manually assign each `appid` to the `k` value. If researchers who take this code are confident with the process of consecutively converting reviews for all apps together, please feel free to modify the `k` value assignments into a loop.

Code Snippet 12 Creating the required review files and folders for SO-CAL

```
k=1
# check whether the folder with a name of current app id exists
folder = str(k)
if not os.path.exists(folder):
    os.mkdir(folder)
cursor.execute("select * from reviews where appid={}".format(k))
result = cursor.fetchall()
for row in result:
    id = row[0]
    text = row[7]
##    sometimes people put reviews in the title field rather than text,
##    then take titles as reviews if texts are empty.
    if text == "":
        text = row[6]
    # output a text file for each review
    outputReviewFileName = str(id)+'.txt'
    outputReviewFile = open(os.path.join(folder, outputReviewFileName), 'w')
    outputReviewFile.write(str(text))
```

```
outputReviewFile.close()
```

- (2) To decide the input and output folders for data pre-processing and modify the `run_text_preprocessing.bat` with them.
- (3) To decide the input and output folders, and any other folders if different with the default, and modify the `run_sentiment_calculator.bat` accordingly. The input folder in this step should be the output folder from the last step, data pre-processing for convenience.
- (4) Copying the text files of reviews produced by code C.5 into the input folder for data pre-processing.

In practice, it might be easier to manage the progress if copying the reviews of one folder (one app) each time. If so, it will be convenient to repeat step 2, 3, 4, 6, and 7 for one app each time.

- (5) In cmd, starting the Stanford CoreNLP server with the command suggested on the SO-CAL GitHub page [64] part 1.
- (6) In another cmd, running the “`run_text_preprocessing`” command from the source code folder where this command is. The results can be checked from the output folder defined for pre-processing.
- (7) Running the “`run_sentiment_calculator`” command. The results can be checked from the database “`tokenizedsentences`” and “`reviews`” tables.

It is worth noting, depending on the Python libraries installed, some computers might complain that “`ModuleNotFoundError: No module named 'unicode'`” or any other modules not found errors when running SO-CAL. If this happens, the errors will disappear after the Python libraries are installed.

In “`run_text_preprocessing`”, there is an issue existing in the path recognition for input and output folders that might not follow conventions. This issue can be bypassed by the example paths below.

Example input folder path:

```
-i '../..../Sample/input/Raw_Text/1'
```

Example output folder path:

```
-o '../..../Sample/output/Preprocessed_Output/1/'
```

Those above two example folders are under a “Sample” folder in parallel to the “Source_Code” folder.

For “run_sentiment_calculator”, the input folder path should be the same with the above output folder path. However, the same folder is read differently here as shown below.

```
-i ../Sample/output/Preprocessed_Output/1/
```

The output folder path:

```
-o '../..../Sample/output/SO_CAL_Output/'
```

It is also worth noting, the paths could vary on computers with different operating systems. Moreover, the last output folder path is not crucial because this framework does not currently fully utilize the full functions of SO-CAL.

3.4 Automatic user requests elicitation

Request Elicitor is a component that elicits user requests from the user review sentences through a set of user requests linguistic rules. This Request Elicitor component elicits only explicit user requests in a qualitative manner. In other words, there are no implicit user requests being mined currently. Moreover, the more comprehensive the linguistic rules are, the more user requests will be mined. The future researchers who take this framework will have more concrete control against which user requests will be mined and how comprehensive the elicitation performance would be.

For example, this is an implicit request:

“Only supports one type of blood sugar meter from one manufacturer.”

This is possibly requesting more than one type of blood sugar meters. However, it is not explicit. Whereas this is an explicit request that this component can elicit with a current linguistic rule in the implementation.

“Some kind of point reward program for challenges would also be great.”

3.4.1 Proposing generalized grammar rules and their trade-offs

None of these linguistic rules is related to domain specific details. The way for the linguistic rules mining in this framework is extracting from occurrences in real user reviews, rather than simply inventing them from keyword combinations. It is assumed that this is the way to create practical linguistic rules therefore increasing performance and decreasing false positive results at the same time.

The two sets of linguistic rules adopt different methodologies. The user requests elicitation linguistic rules are created from different real scenarios and subsumed into as few rules as possible. In the implementation submitted with this thesis, there are only 89 implemented linguistic rules for user requests. The mechanism of user request linguistic rules is more complicated than the topic-opinion pair rules. The larger the number of the rules is, the more difficult the management would be. Therefore, considerable efforts have been invested to find the trade-offs between the number of linguistic rules and the coverages of certain linguistic rules. An example is presented in the section 3.4.3 later.

3.4.2 Tackling the data sparsity problem

The data sparsity problem in the linguistic rule's extraction process is a dominating problem and is more severe for user request elicitation linguistic rules. In order to overcome this problem, the process started from 1000 user reviews first in the manual analysis, but actually ended up with manual analysis of 6081 user reviews (11563 sentences). This could be sufficient for the topic-opinion pairs, but still not enough for the user request elicitation rules.

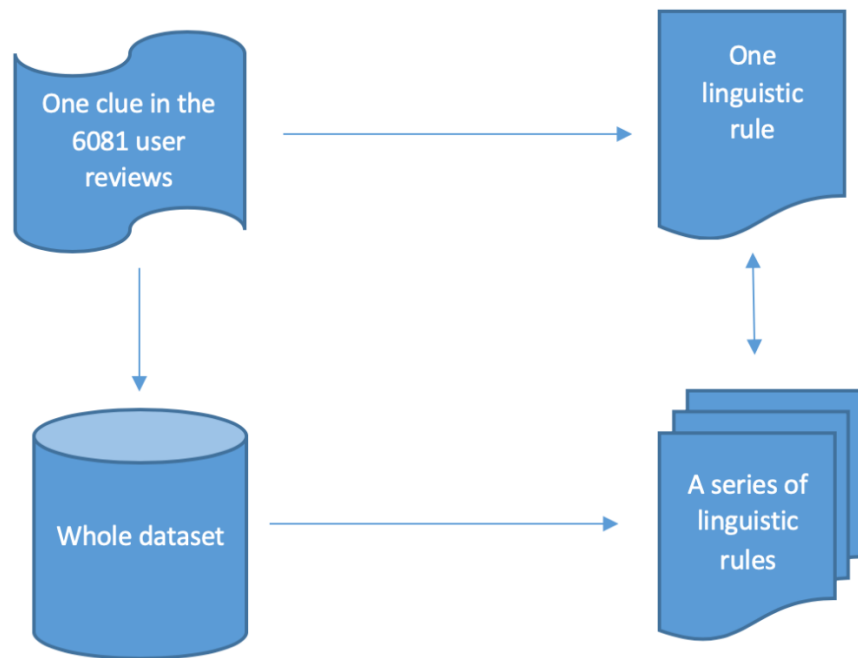


Figure 10 User requests linguistic rules manual mining process

Therefore, an approach is proposed as depicted in Figure 10 to tackle the data sparsity problem further. The 6081 user reviews are from one mobile app. The dataset one where that mobile app is from contains 21280 user reviews in total. For any sentence that contains a user request in those 6081 user reviews, it is treated as a clue, namely an opportunity for a user request linguistic rule from the 21280 user reviews. The possible keyword combinations or ones derived from similar keywords were queried through the whole dataset in the database. Hence one clue could be taken as a seed to create a set of linguistic rules based on the whole dataset appropriately. Sometimes when any keyword combination was not good enough to

form a linguistic rule, it should be broken down further to more detailed keyword combinations, until more linguistic rules were identified with satisfactory performance. Consequently, this approach could tackle the data sparsity problem as best as possible. This process is explained in detail in the example below.

3.4.3 An example process of mining user requests linguistic rules

How to balance the trade-off between the linguistic rules number and coverage and how to tackle the data sparsity problem can be illustrated in an example process of mining a set of user request linguistic rules in this subsection. In this example, the starting clue is this sentence:

“Miss seeing Lily's antics when I enter a reading or food, and **can you add** half a second before she starts?”

From this sentence, “can you add” is a set of keywords that can help to elicit the request “half a second before she starts”. Searching this clue “can you add” against the whole 21280 user reviews, gives 4 sentences containing this set of keywords, and all the 4 sentences are explicitly expressing user requests. However, a search for other keyword combinations derived from this clue should be carried out against the whole dataset.

The search can expand to all possible keyword combinations or ones derived from similar keywords. These keyword combinations could be “can you”, “could you”, “would you”, “can I”, “could I”, “would I”. All these keywords can match a pattern: a modal verb + a personal pronoun + a base form verb.

However, inventing keyword combinations is not a good way to create linguistic rules in practice. These combinations have to be verified through the real dataset. The verification results on the 21280 user reviews in dataset one are below. For each keyword combination, a search is performed on dataset one to find how many times they occur. Among the occurrences, percentages of containing user requests are calculated and reported in Table 17.

Table 17 Initial keyword combinations and their occurrences in dataset one

Keyword combinations	Number of occurrences	Percentage of containing user requests
could you + <VB>	11	91%
could I + <VB>	0	0%
would you + <VB>	4	50%
would I + <VB>	2	0%
can you + <VB>	27	59%
can I + <VB>	22	50%

It is obvious that the two combinations that have 0 % user request should be ruled out. But none of the rest of combinations has good enough precision to become a linguistic rule as it is. Therefore, the author took the occurrences of each of the rest combinations for further breaking down. Taking the example of the 11 occurrences of the first combination, “could you + <VB>”, the data are analysed below.

Table 18 Request analysis for “could you + <VB>”

reviewId	appId	text	Request / Not
6625	4	It would be great if you could use the fool database for FatSecret, as it has food entries specific to various countries. It would also be great if the app could access the food data from Google Fit, as you can use other apps to write data to it. Also could you guys look into adding calories burned with exercise please. Including the calories burned from apps such as Google Fit Then the app would get 5 stars. PS I am a paying subscriber	Request
7484	4	Very useful app. Could you please add Basaglar?	Request
8222	4	Could you add Persian language please?	Request

8338	4	Could you install a sick explanation for high numbers? I have been very ill and my numbers were High. I did leave notes, but think it would be clearer to tab sick into my log with the numbers. Thanks	Request
9821	5	This is a great app but 2 things are missing and they are after dinner and before bed could you please add these	Request
10053	5	Easy to use and meets all my requirements. My Wish List: When you export the dates, could you put them in yyyymmdd format please - or at least an option. Not everyone in the world uses mm/dd/yyyy and it can be quite confusing explaining it to a non-techie doctor.	Request
10422	5	This app is great for me. I'm a newly diagnosed diabetic and it offers everything I need. Granted I use it for just my glucose level, the app itself has numerous other uses. From a food diary to the ability to record blood pressure the app is great! With all this and it's free, how could you need something else?	Not
16270	6	Hello the app looks good. Haven't had a chance to really try it out. Could you please add the feature of changing the language to spanish???? Thanks	Request
20012	17	I have been using ontrack for more than a year now and have nothing but good things to say. I email and print graphs for my doctor, he has come to expect the reports. My doctor also recommends OnTrack to his patients. And free too! What else could you want....	Not
20793	17	Could you add a bmi calculation? Then I don't need another program any more ... the graphs seems a bit compressed too, but I love your program.	Request

21051	18	Hi, could you please fix the bug which causes the Previous Readings at Blood Glucose to appear distorted and abnormally large, probably because of the SHARE icon? Sony Xperia M	Not
-------	----	--	-----

From these 11 occurrences in Table 18, “could you + <VB>” can be subsumed into the following rule.

Table 19 A sample rule for “could you + <VB>”

(how What else) Could could you (guys) (please) add put install look (into) <request>

By explanation, this rule is looking for “Could you” or “could you” followed by a base form verb “add”, “put”, “install” or “look into”. Moreover, “guys” or “please” can optionally appear before the base form verb. But there must not be “how” or “What else” appearing before “Could you” or “could you”. Then the text after the base form verb is the user request to elicit according to this rule. At the time of writing, this rule has 100 % precision against dataset one.

However, the author was trying to group user request linguistic rules as much as possible and balance the trade-offs between the number of rules and the performance of specific rules. After the consideration for the trade-off, the author decided to subsume the resulting linguistic rules of three keyword combinations, “can you + <VB>”, “could you + <VB>”, and “would you + <VB>”, into one rule (77), but leave “can I + <VB>” to another rule (78). Therefore, this process eventually produced two rules that cover occurrences of 44 and 22 in the dataset one respectively.

Table 20 One clue leads to two linguistic rules covering 66 occurrences in dataset one

77	(how nor why what where what more what else how often not only no longer) can could would you (guys) (please) add put make give input change create install consider look (into) <request>
78	can I i have + <NN NNS NNP NNPS> -> <request>

The ways of coding these two rules are introduced next.

3.4.4 Single rule structures and flexibilities.

To enable a single rule to work, a training text from NLTK corpus has to be decided for the tokenizer to produce consistent patterns. In the first implementation of this framework, it is set as

```
train_text = state_union.raw("2005-GWBush.txt")
```

and it remains the same for both the debugging code and the actual elicitation code that talks to the database.

A typical user request linguistic rule is composed of two parts: chunking (with chinking if necessary) and regular expression. Rule 78 in the example above can be expressed as below for the chunking and chinking grammar.

Code Snippet 13 An example chunking and chinking grammar

```
chunkGram = r"""\Chunk:(<MD><PRP|VB|NN><VBP><.*>*<NN|NNS|NNP|NNPS><.*>*)  
}<\.|\?>+{"""
```

This grammar is looking for a modal verb, followed by either a personal pronoun, or a base form verb, or a noun, then followed by a non-third person singular present verb, then anything for any time of repetitions before and after an any form of noun, and this grammar must end before the end of sentence punctuation, such as period, exclamation mark or question mark.

The reason for <PRP|VB|NN> to represent either a capital “I” or a lowercase “i” in this grammar is that in this implementation, the text of the sentence is converted to lowercase. The impact of the .lower() is that the capital “I” is parsed as “NN” rather than “PRP”, and sometimes could be “VB”, when it is changed to lowercase “i”.

The period and question mark in the line of chinking grammar in Code Snippet 13 have to be escaped with a backslash. Otherwise, a period means an identifier for any character, except for a new line, whereas a question mark means a modifier matching 0 or 1 repetitions.

Only texts that have passed the chunking (with chinking) grammar can be tested against the regular expression further. In this example rule 78, the regular expression is below.

Code Snippet 14 An example regular expression test

```
regulars = re.findall(r'[c|C]an\s[i|I]\shave\s(.+)', str(string))
```

In this example, the code is testing against the text for either a “can” or a “Can” followed by a space, then either a lowercase “i” or a capital “I”, a space, a “have”, and a space. If the text passes this test, then the text after that is returned as the resulting user request.

But the resulting user request must not be empty in this example above. “(.+)” means that anything that must appear at least once in this pair of parentheses is the target text.

Rule 78 is a typical rule in the implementation of this component. This implementation defines a function for each rule and returns the user request value. In the function, there are normally two tests, one for chunking (with chinking) grammar, and one for regular expression.

However, rule 77 above needs one more test to ensure that the texts that pass the regular expression test must not begin with anything among “how | nor | why | what | where | what more | what else | how often | not only | no longer”.

This can be achieved by the code in Code Snippet 15.

In this example taken from rule 77, the regular expression defines two pairs of parentheses, which are represented by request[0] and request[1] respectively. Inside the first pair of parentheses, there are definitions for all the terms that must not appear before a user request that matches the chunking grammar and the regular expression in this rule. The modifier of this pair of parentheses is “*”, therefore the terms can appear 0 or any times. The second pair of parentheses is actually the target text for the user request. The logic below the regular expression definition prohibits the return of the target text if any term defined in request[0] does appear. This is an example of the flexibilities that the current user request elicitation rule structure can fulfil.

Each user request linguistic rule is implemented as a function currently. Each function contains a “try” block and an “except” block. It is very important for readers to be aware that the “except” block in each function cannot use “pass” here. The reason for that is there are many rule functions with similar structures in parallel in the code. In Python, “pass” in an “except” block actually means passing the last variable’s value on to the next available one. This can create unexpected results if this is not aware of.

Code Snippet 15 An example of flexibility in current user request rule structure

```

regulars = \
re.findall(r'([h|H|n|w|W][o|h][w|r|y|a|e|t][t|r]?[e]?[s][o|m|e|l][f|o|l|n][t|r|s|l|n][e|y|g][n|e]?[r]?[s])*[c|C|w|W][a|o][n|u][l]?[d]?[s]you[s][g]?[u]?[y]?[s]?[s]?[p]?[l]?[e]?[a]?[s]?[e]?[s]?[a|m|g|i|c|p|l][d|a|i|n|h|r|u|o][d|k|v|p|a|e|t|s|o|n][e|u|n|a|t|k|s]?[t|g|a|i]?[e|l|d]?[l|e]?[r]?[s]?[i]?[n]?[t]?[o]?[s]?[s](.+)', str(string))

for request in regulars:
    if request[0] not in ['How often ', 'how often ', 'How ', 'how ', 'nor ', 'why ', 'Why ', 'what ', 'What ', 'where ', 'Where ', 'What more ', 'what more ', 'What else ', 'what else ', 'not only ', 'Not only ', 'no longer ', 'No longer ']:
        return request[1]
    else:
        return "None"

```

For the full code of all linguistic rules implemented in this submission, readers are referred to the list of code C.8.

For the POS tag list and the convention of chunking (chinking) grammar and regular expressions, readers are referred to Appendix C POS tag list and user request rule convention.

When writing reviews, users sometimes use upper case words in the reviews. Those words can be parsed differently in both POS tags and dependency trees. In order to eliminate such impacts, all sentences are converted into lower cases at the later stage of this implementation, which applies to this component Automatic user requests elicitation and the next component Topic-Opinion extraction. The performance of this framework is therefore improved by

eliminating the false negatives that could match a rule. This also improves the clarity in the ontology by reducing the number of different individuals.

The most obvious impact of “.lower()” is that the POS tag of capital “I” changes from “PRP” to “NN” and sometimes others. There could be other impacts, such as ('Android', 'NNP') becoming ('android', 'JJ').

In the current implementation of linguistic rules, both normal sentences and lower case sentences are accommodated. If the researchers who take this framework in the future decide to remove the “.lower()”, these two sets of linguistic rules in this component and the next one will still work with expected performance. Although this is not recommended because the especially on purpose uppercase words will flee from the rules.

3.4.5 The current coding structure for user request linguistic rules.

The current coding structure for user request linguistic rules is depicted in Algorithm 2. Each linguistic rule is defined as a function in this implementation. Because each rule is a standalone function, it is flexible in its own structure and functionalities. It can contain not only chunking (and chunking if necessary) and regular expressions, but also any further string operations. The previous example rule 77 defines two target texts in the regular expression, then the test of non-existence of one of the targets decides the return of the other. It is also practical to do more string operations for the resulting requests before their returns. Such operations can include .replace() in current rule 96, tests if a string starts with another string (rule 96), tests for a string “in” or “not in” the request string (rule 108), or adding a prefix (rule 90). Even more, if it is certain that a rule is robust enough without the necessity to define a chunking grammar or a regular expression, it is OK to just elicit the user requests according to a string test. For example, in the current rule 93, there is no regular expression defined.

Algorithm 2 Automatic user request elicitation

Input: A sentence in the user reviews

Output: If this sentence matches a request elicitation rule, the elicited request will be output into the database, otherwise nothing is outputted.

```

set rule = 0 and request = "None";
## Every rule has a keyword test, for example:
if (request == "None") and (a review sentence contains keywords for rule X) then
    rule = X;
    execute function (rule = X);
    if the review sentence matches the rule in the function (rule = X) then
        request = "REQUEST";
    else
        request = "None";
    end if
end if
## Keyword tests for other rules are omitted, until all rules have been tested.

if rule != 0 and request != "None" then
    output request.
end if

```

All rules are behind a switch. Python does not actually have switch-case statements. The switch logic is achieved via a `numbers_to_rules(rule)` function. For its detailed definition, readers are referred to the list of code C.8 User request elicitation code.

Each rule has a corresponding keyword test before the switch. If a sentence passes the keyword test, it will enter the rule function through the switch for user request elicitation. The elicitation could pass, and could fail. If it passes, the variable "request" gets a value. If it fails, "request" remains the initial value "None". By keeping the initial value "None" for "request", the failed sentence can smoothly enter next keyword tests until a user request is found.

Explained further, the reason to set the initial "request" and the failed "request" values to "None" is that a sentence can go through all possible keyword tests and therefore enter all possible rules until the "None" value changes to a user request in the first matching rule

function. When a user request is found, the “request” value is not “None” anymore, therefore all the rest of keyword tests will stop. This purpose is achieved by the initial “None” request value, the “None” values for all failed “request”, and the test of “if request == "None"” in all the keyword tests.

If the current logic of the “None” request values isn’t implemented as shown in Algorithm 2, a sentence could enter a rule but lost due to the failure of matching this rule, or a sentence could enter multiple rules, which costs more computing resources, and is hard to control what the resulting request would be.

In this linguistic rule structure, the sequence of the rules matters. Therefore, it is recommended that the more specific the rule keyword test is, the earlier the rule should position in the sequence. In order to facilitate future researchers to manage and enrich further linguistic rules, a spreadsheet “UserRequestLinguisticRules.xlsx” containing key details of all rules is provided in the list of code folder with this submission.

3.4.6 Two example rules inspired by English grammar

Not all linguistic rules have to go through the example process introduced in 3.4.3. Future researchers can design their own strategy for rules, especially when replacing this set of rules with a new set of rules for a new research question.

In this set of rules in the deliverables, there are some rules that are standalone or inspired by English grammar. An example is the rule 100 below.

Table 21 Rule 100: wish + past tense simple verb or a modal verb

Rule definition for human	wish + VBD MD → <request>
chunkGram	r""""Chunk:{<VB VBD VBP JJ><.*>+<VBD MD><.*>+} }<\. \?>+{""""
Regular expression	r'[w W]ish\s(.+)'

Rule 100 is inspired by the English grammar when people express a wish. It is relatively certain in English that if a word “wish” is followed by a past tense simple verb, people want

a situation in the present (or future) to be different. In the case of a user review, this is regarded as a user requesting something to be different.

It is also relatively certain but a little less confident than the example above in English that people request something would be different if the word “wish” is followed by a modal verb.

Rule 100 merges these two cases together.

In rule 100, although there is only one keyword “wish” in the regular expression, it is able to control the POS tags of the verb that follows the word “wish” by the chunkGram. This is a major benefit of combining chunkGram with regular expression in the linguistic rule definition.

Such a benefit also enables rule 87 to work. The definition of rule 87 is in Table 22.

In rule 87, if a keyword “maybe” appears as an adverb, and is followed by a phrase or a clause containing a verb, a past tense simple verb, or a non-third person present tense singular verb, or an adjective, this phrase or clause is highly likely a user request.

Similar to rule 100, in rule 87, there is only one keyword in the regular expression as well. But the chunkGram provides four varieties for the following phrase or clause to be identified as a user request.

Table 22 Rule 87: Maybe + a verb or an adjective

Rule definition for human	maybe <VB VBD VBP JJ> → <request>
chunkGram	r""""Chunk:<RB><VB VBD VBP JJ><.*>+} }<.\ \?>+{""""
Regular expression	r'[m M]aybe\s(.+)'

3.4.7 The suggested way to mine the rules and enrich them in the future.

Future researchers are recommended to take the similar approach as introduced in 3.4.3 An example process of mining user requests linguistic rules, or create linguistic rules inspired by grammar or practical findings. Once the linguistic rules have been decided as in Table 20 One clue leads to two linguistic rules covering 66 occurrences in dataset one, or a linguistic rule is needed to update, the following code and spreadsheet will be helpful.

(1) The code C.6 Code to parse a sentence into Part-Of-Speech tags.

This piece of code produces the POS tag for each token. The suffix `.lower()` at the end of the variable value of “`sample_text`” helps produce POS tags for a lower case sentence. Researchers can take `.lower()` as a switch between results of normal sentences and lower case sentences.

It is recommended to stick to the same `train_text` for all three pieces of code in this component.

(2) The spreadsheet `UserRequestLinguisticRules.xlsx` in the list of code folder.

This spreadsheet lists all the current user request linguistic rules in the sequence of that in the code C.8. Each row of a rule record has a rule number, chunking (with chinking) grammar, regular expression, the name of the request variable returned, keyword test, and the linguistic rule definition for humans to easily understand.

It is strongly recommended that future researchers please keep the rule numbers stable. The rule numbers in this spreadsheet are the same with the rule numbers in the debugging code C.7 and the actual code C.8. It would be helpful for new rules to be managed in this spreadsheet in the same way. The reason for that is the more the rules are, the more difficult it is to manage them with brains.

(3) The code C.7 Debugging code to help user request linguistic rules’ creation.

This debugging code produces results for researchers instantly in the rule coding process. Depending on the sentences that the new rule accepts are normal or lower case, an assignment to the variable “`exampleString1`” is needed with either a sentence string, or “`the sentence`”.`lower()`. Then a new rule needs to be pasted below other rules, but before the “`except`” block.

Modifications are needed for the “`chunkGram`”, the “`regulars`”, and the rule numbers in the print statement and the description comment above this new rule. The code C.6 is especially helpful when defining the “`chunkGram`”.

When a rule subsumes multiple variants, it is necessary to test each variant with sufficient example sentences. This is an important step for defining the chunking grammar.

When testing a rule with an example sentence, in the output window of the Python shell, only the printed strings of the target user requests represent the rules that the sentence has passed. The rule numbers, which are printed only without the company of a string, are the rules that have matching chunking grammars only.

This debugging code is helpful to decide the sequence of the rules when a sentence matches multiple rules with different outputs.

(4) The code C.8 User request elicitation code.

When a new rule is successfully defined and tested through the debugging code above, it needs to be adapted to the code C.8 User request elicitation code, which will work with the database. There are three places in the code that need to be adapted.

The first is the rule function. The structure is slightly different with the debugging code. But the rule number, the “chunkGram”, the “regulars” are the same.

The second is the entry in the “numbers_to_rules(rule)” function.

The last place is the keyword test. The keyword test’s design is important if the rule itself is complicated. This is also the place to implement the sequence of the rules.

3.4.8 Attention that might save time for future researchers.

Always rely on code C.6 to parse a sentence, and never predict POS tags by grammar or experiences. When a sentence cannot pass a rule in the debugging code for an unknown reason, carefully compare the POS tags between the output from code C.6 and the “chunkGram” definition in the rule, word by word. Then the unknown reason will frequently be fixed.

The space “\s” is important in the definition of regular expressions, because the string inputs are parsed with all tokens separated with a space that doesn’t appear in the original sentences somewhere if the words are concatenated previously. For example, “doesn’t” becomes “does n’t”, which is expressed as “does\s n’t” in the regular expression.

In the debugging code there is a commented “print (string)” above the regular expression statement in each rule. This is helpful when it is not certain why the regular expression does not work.

Don’t forget to escape special characters whenever they are used, such as “PRP\$” should be “PRP\\$”.

3.5 Topic-Opinion extraction

This topic-opinion extractor component relies on Stanford Dependency Parser and dependency tree traversal to extract topic-opinion pairs, and then filter them according to the ontology subjects' definitions in order to map them onto the ontology classifications. The process can be depicted in the figure below.

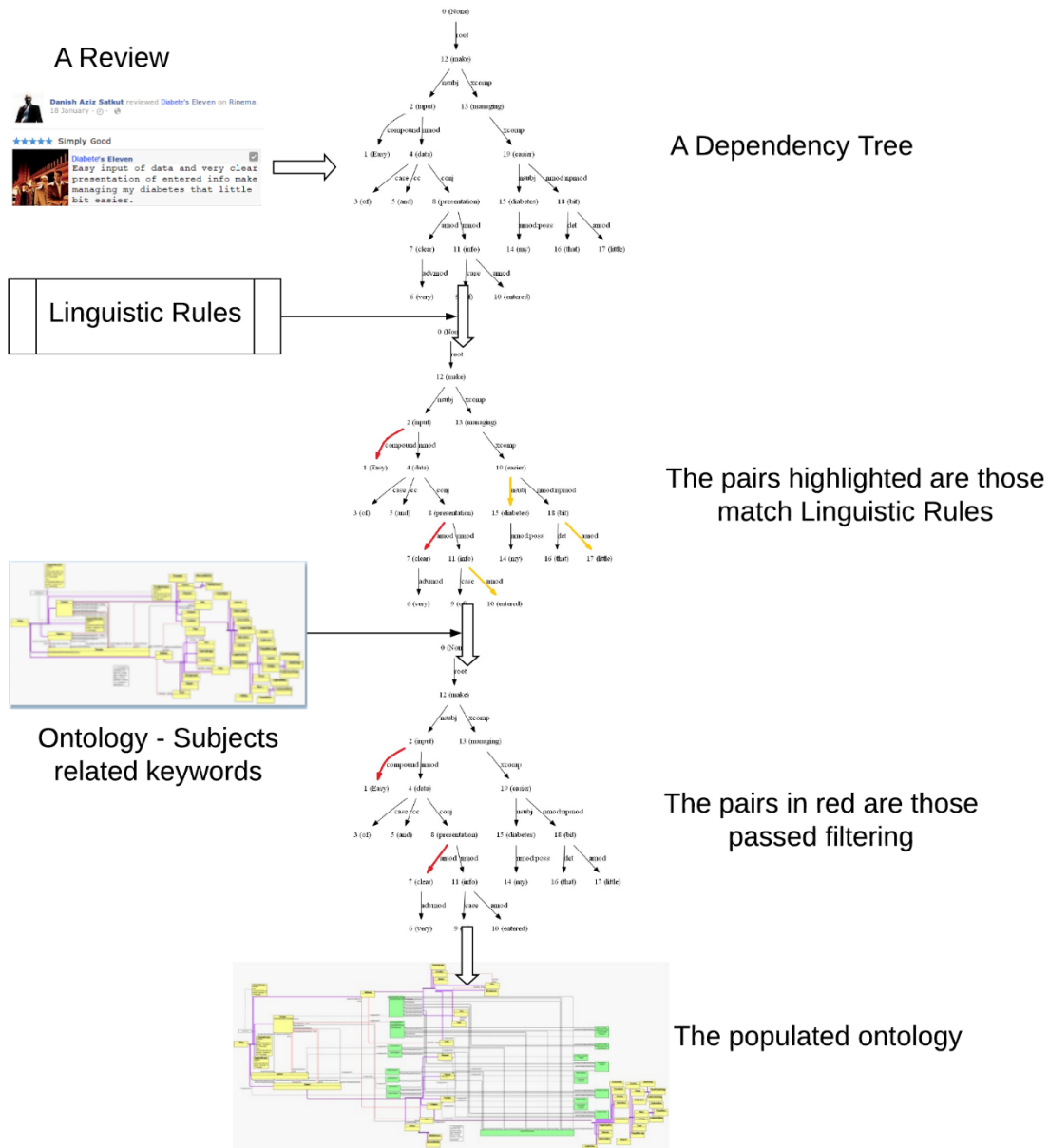


Figure 11 Topic-Opinion extraction process.

In this section, an example will illustrate the topic-opinion extraction process on a dependency tree in subsection 3.5.1. Then subsection 3.5.2 and 3.5.6 will expand on how this component works in more detail in two steps: traversal on the dependency trees and keyword mapping to the ontology next. The first step, traversal of the dependency trees, has two versions of algorithms that are introduced in subsection 3.5.4 and 3.5.5 respectively. Subsection 3.5.3 describes how the negation words are handled. The next subsection 3.5.7 explains how this component integrates with the previous component 3.4 Automatic user requests elicitation. Subsection 3.5.8 suggests adjustments that future researchers may adopt in the current structure, and provides a piece of debugging code to help with the adjustments. The last subsection 3.5.9 introduces the efforts that this component has made towards the implication handling when users might express opinions to the current mobile app.

This component requires Stanford Parser, WordNet, and graphviz (helping the debugging). Readers are referred to Appendix A Software installation for necessary help on software installations.

3.5.1 A simple example for how this component works for a dependency tree

The process of how this component works for a dependency tree can be illustrated in a simplified example. The user review sentence in this example is below:

“Easy input of data and very clear presentation of entered into make managing my diabetes that little bit easier.”

Figure 12 is the dependency tree that Stanford Dependency Parser produces for this sentence. From this tree, it is evident that opinions are linked with their topics via a single arc. This component only deals with such direct relationships between topics and opinions. The process that informs this decision is reflected in the previous Opinion mining section in the Literature review chapter.

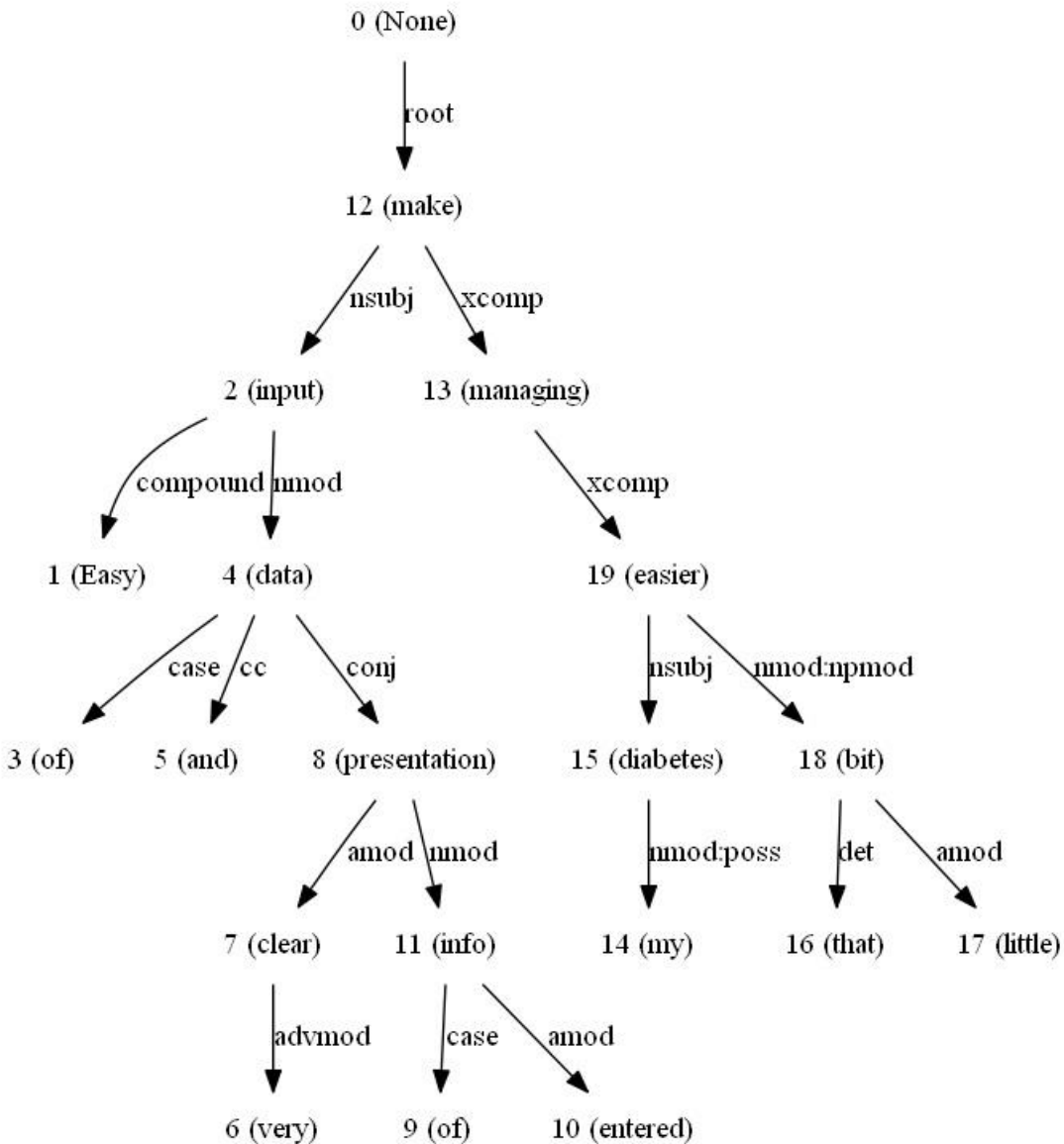


Figure 12 A dependency tree

These arcs are highlighted in colours in Figure 13. Among these arcs, some are interesting for stakeholders, namely they are the opinions that users have against specific topics about the current mobile app. These are the user feedback on existing requirements that stakeholders aim to elicit. They are “Easy – input”, “clear – presentation”, and “easier – managing” on this tree, which are highlighted in the red colour. But the rest arcs are trivial, which are highlighted in the yellow colour. The yellow arcs might be topic and opinion pairs, but they are not the targets in this requirement elicitation process.

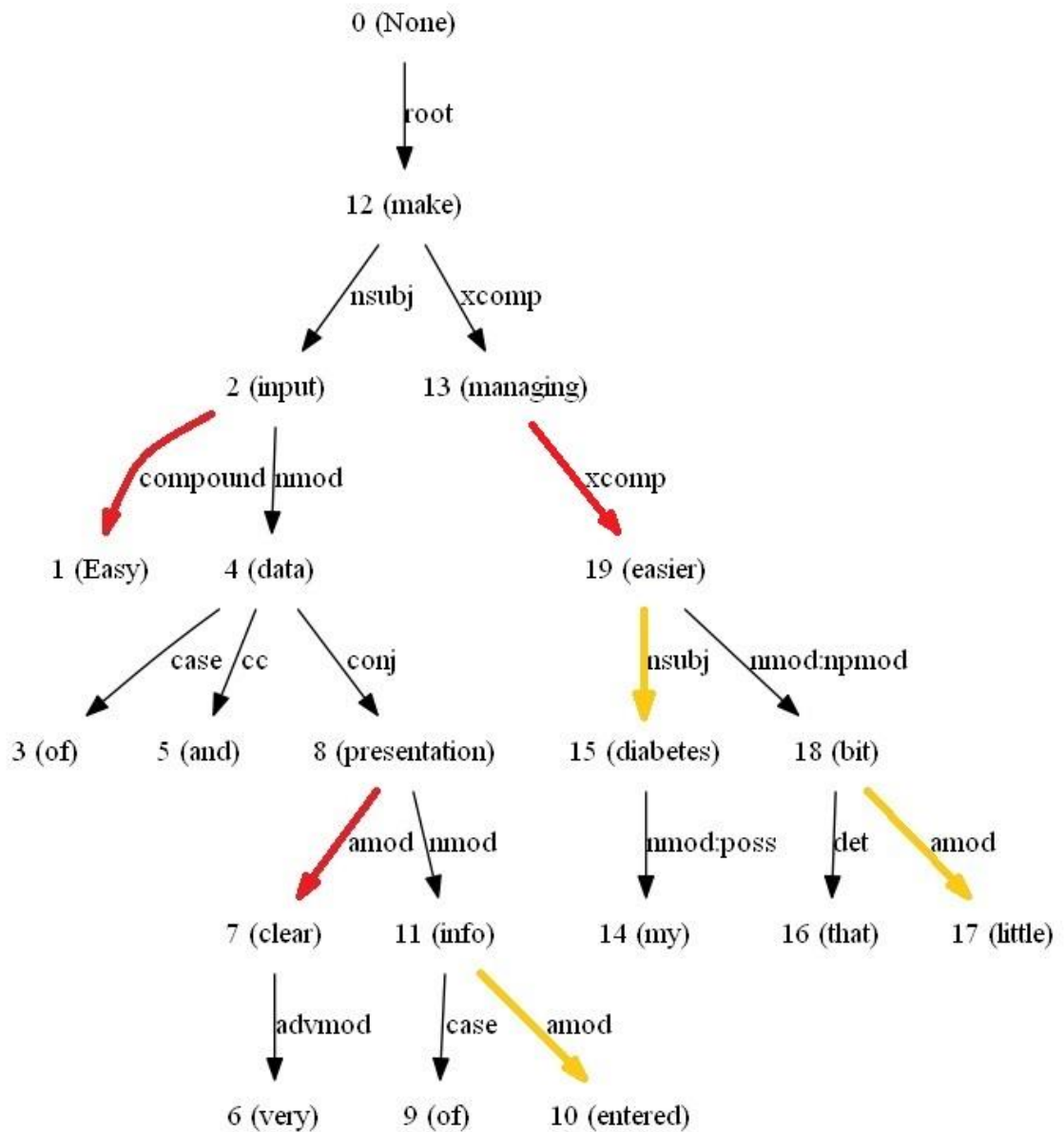


Figure 13 A dependency tree with arcs that match linguistic rules highlighted

The purpose of this component is to help stakeholders and requirement engineers distinguish the arcs, output the interesting arcs only, and grant them the capability of twisting the standard for qualified arcs according to their needs. Figure 14 illustrates the resulting arcs in red on the tree.

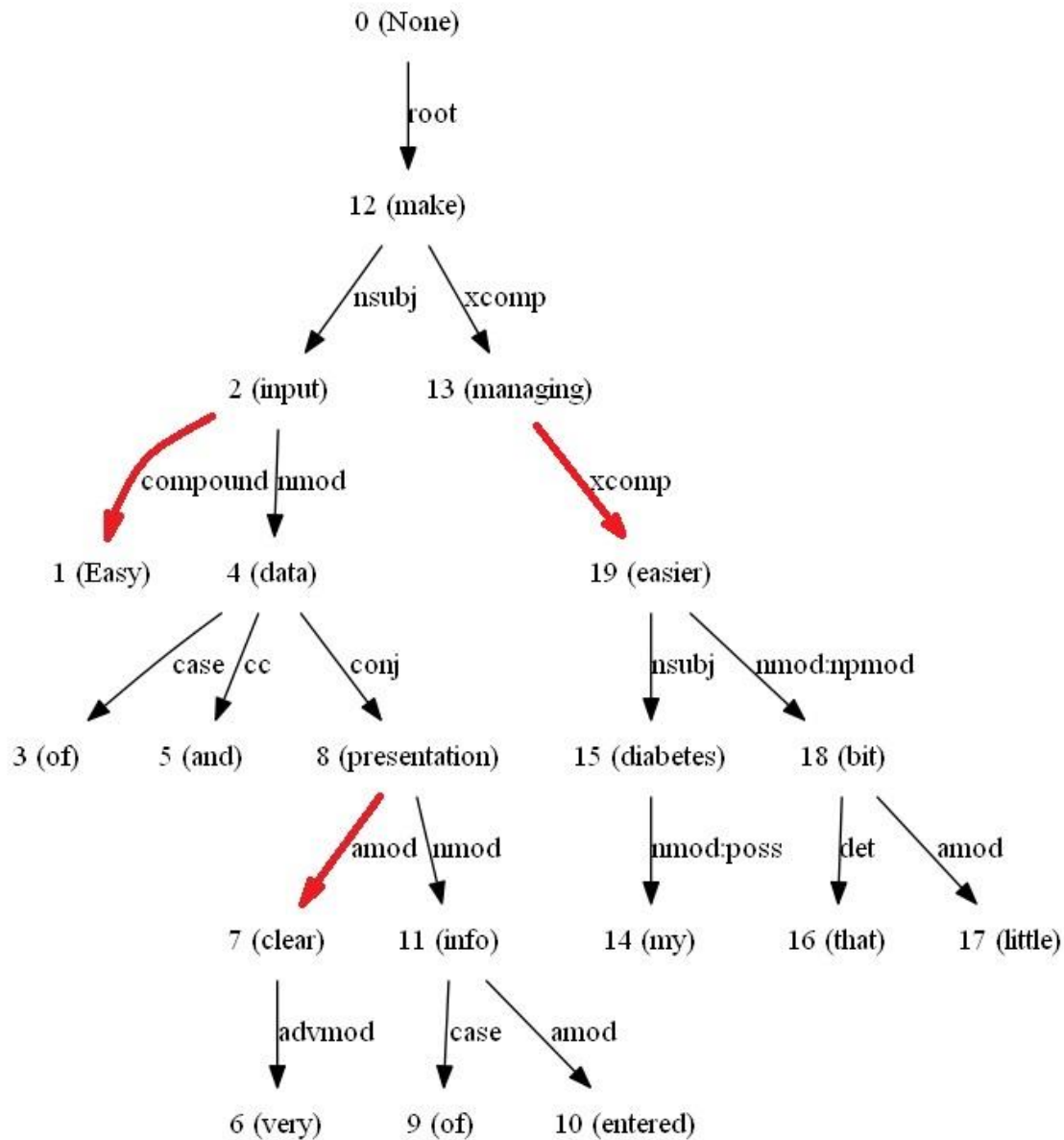


Figure 14 A dependency tree with the arcs that pass filtering highlighted

This component is composed of two steps. Step one traverses the dependency tree and extracts topic-opinion pairs that satisfy a number of linguistic rule definitions, which is illustrated from Figure 12 to Figure 13. Step two filters the produced topic-opinion pairs according to stakeholders' interests, which is presented from Figure 13 to Figure 14. This is achieved via comparisons between the topics and a keyword array that contains the keywords for the interesting topics.

The next subsections introduce how this component works in detail.

3.5.2 Topic-Opinion pair extraction via linguistic rules

In this first step, the task is traversing the dependency tree and comparing every arc with the linguistic rules in order to find the matched arcs.

There are some questions / problems that need to be resolved in this step.

Q1. Stanford Dependency Parser parses every sentence into a dependency tree, therefore a data structure that can represent this tree is needed in this framework in order to have full access to all the information from this tree.

Q2. The dependency tree is not always regular in the structure. For example, the addresses of the nodes are not always consecutive. This has to be handled properly in order to get the exact structure of the tree.

Q3. Moreover, sometimes a dictionary of multiple nodes appears as one node in the position of one address. Thereby, the way of building the tree must be able to interpret multiple nodes as children of the node from the dictionary when they appear as one.

Q4. Stanford Dependency Parser parses different sentences into different trees that are different in sizes and structures. Therefore, a method is needed to traverse the different trees in order to find all the arcs that link a topic and an opinion together.

Q5. This method should be able to pick up negation words if Stanford Parser parses negations correctly.

Q6. The linguistic rules must be presented in a neat and easy to manage way.

The solutions for above questions are explained below.

For the question in Q1, the data structure is achieved by a class “Node”. The class “Node” accepts a dictionary as the input parameter type because Stanford Parser produces dependency trees as dictionaries. All nodes on the tree are read in turn and constructed as an object of the class “Node”. The “__init__” method of class “Node” initiates the class with corresponding values from the input node. After a node is set up via the “__init__” method, another method “add_child” is necessary to build the tree. The “add_child” method accepts

a node as the parent, and a node as the child. This method simply appends this child to the parent's children after a simple test of the child's type if the child is also a dictionary, which every node must be.

When parsing a dependency tree, Stanford Parser assigns an address number to each node. However, for some reasons, the addresses are not always consecutive. One possible reason is the punctuation in the middle of the sentences because all punctuation is ruled out from the trees. In order to deal with this question in Q2 properly, a loop is combined with a node list as implemented in the code C.9. The loop loops through all the nodes in the result of a dependency tree. A node list is created inside this loop as a local variable. Before appending a node to this node list, this node must pass a test against its address number. If its address number equals to the index of the node list, this node is appended to the node list under the index. Otherwise, an empty string is appended to this node list to occupy that index.

When the question in Q3 happens, multiple nodes appear as one node. This phenomenon causes the later nodes to be cut, therefore the branches are missing for further processing. The solution to question Q3 is realized in the process of building the tree. When the node list is completely populated, it is used to build the tree. The first node with the index of "0" is taken as the root. Then each node is appended to its parent. The relationship of parent and children is read from the dictionary value of "deps" of the parent node. This is the value where sometimes multiple children appear as one node. When this happens, the children nodes' addresses are separated by commas when wrapped into a pair of square brackets, for example "[9,10,12]". They are firstly split by the square brackets and commas, so they are taken out and separated. Then within a loop, each of the children is added to the current parent node through the "add_child" method of the "Node" class.

In the "Node" class, another method "find_related_pairs_withNegation_withRules" is created to tackle the problems in Q4, Q5, and Q6.

To resolve Q4, this method is designed as a recursive method, which will traverse all nodes on a dependency tree. This method accepts a node, a "related_pairs" list, and a node list as parameters. When this method is called in the main code of C.9, the process of building the tree has been completed. Thereby, the input node is the root of the tree, namely the node list

index “0”, and the input node list is the completed tree. The “related_pairs” is an empty list that is created before this method is called. The reason for creating this empty list before the method call is that this method is a recursion, which will recursively execute every line of code in the base function. If this list is created inside the method, the list will be never completed as it is expected.

The base function of this method is for each arc on the tree, to identify the parent, the child, the negation, and the relationship of this arc, then compare with each linguistic rule, and output each matching case to the “related_pairs” list. The recursion part of this method repeats the base function to each of the children (if exist) of the node. The “related_pairs” list is finally returned.

The question of negation words handling in Q5 is resolved in the base form of this method. There are two versions of this Topic-Opinion Extractor code in this thesis. The logics of the negation handling in these two versions are different. In version one, the logic is rather basic. Besides identifying the addresses of the parent and the child, this method in version one goes up to the parent node, then looks down for all children of the parent node, in order to look for any child (the original node’s siblings) that has a value of “neg” (meaning “negation”) for the “rel” (meaning “relation”). If such a child is found, this child node’s address is assigned to the negation value. Otherwise, the negation value is a negative number, which is never possible to get from a dependency tree. In version two of the Topic-Opinion Extractor code, the negation words handling is enhanced along with the overall logic that is also more complex. This is explained in more detail in the next subsection 3.5.3 Dealing with negation words.

The number of linguistic rules in version two of this component is 121. For the 121 existing linguistic rules, readers are referred to Appendix D Topic-Opinion pair extraction linguistic rules. They are also in the code C.10 Topic-Opinion pair extraction via linguistic rules Version two. The linguistic rules should be further enriched when adapting to new domains.

It will make the method “find_related_pairs_withNegation_withRules” cumbersome and difficult to manage if coding all these linguistic rules completely inside it. Therefore, the

solution to question Q6 is important. In the current implementation, it is achieved by the Open-Closed Principle (OCP): open for extension, but closed for modification.

The current linguistic rules are managed in a two-dimensional array. Each line of the array represents a rule with the rule number, relationship, POS tag of the child, POS tag of the parent, and who is the subject topic (the rest one is the opinion).

An example rule is “[49, 'amod', 'JJ', 'NN', 'parent']”, which is one of the rules matched in the previous example figure in 3.5.1. In this example, the rule number is 49. The relationship between the child and the parent is an adjectival modifier. The child is an adjective “clear”, and the parent is a noun “presentation”. The subject topic is the parent “presentation”, thereby the opinion is the child “clear”. The current rules are managed in alphabetical order in the array. The future researchers can add and manage more rules just through this array, which is convenient and benefited from the Open-Closed Principle.

The recognition of each rule especially for who is the subject topic is achieved within the method “find_related_pairs_withNegation_withRules”. This method loops through the rules looking for the rules that have the same relationship, child tag, and the parent tag. Then the last subject topic value test decides how to output this topic-opinion pair, namely in which order (who is the topic, and who is the opinion).

For the full logic and the processes of how the above 6 problems are resolved, readers are referred to the list of code C.9 (version one) and C.10 (version two).

For the detailed definitions of the relationships in the rules, readers are referred to Universal Dependencies webpage [109].

In this submission, there are two versions of the traversal function for Topic-Opinion extraction. Version one is simpler in logic and suitable as a starting point of a research question for the popular data discovery purpose. Version two provides better performance than version one due to the usage of a set of popular opinions, picking up the orphan opinions and the enhanced logic. Orphan opinions are opinions that appear as having no topics associated. They are explained in more detail in subsection 3.5.5. The advantages and disadvantages for both versions are discussed in more detail in the Discussion chapter. In this section, the structures of both versions are reported.

The next subsection introduces the current negation words recognition method. The latter two subsections describe the two versions of traversal function respectively.

3.5.3 Dealing with negation words

Both versions of the traversal function need to deal with negations. Most of the time, negations appear with a relationship of “neg” associated with the opinion word. It could appear as the parent, a sibling, or a child. However, sometime, negation words appear with the relationship of “dep” or others, such as “nmod”, “advmod”.

In this prototype, currently negation words are dealt with in the three situations below. It is a very interesting topic for an optimized solution of dealing with negations in the future research.

In version one of the traversal function, only the second scenario below, negation appears as a sibling of the current node, is considered.

In version two of the traversal function, all three scenarios are considered. When the current node is seeking the pairing relationship tests with its parent, scenarios 2 and 3 are considered. Namely, only situations of the negation word appearing as a sibling of the current node, and the negation is its parent, are considered. When the current node is seeking the pairing relationship tests with its children, scenarios 1 and 3 are considered explicitly. The second scenario, if it is true, is automatically inherited from the assignment above the code.

The negation words handling in version two of the traversal function performs better than that in version one. Section 4.5 in Results and evaluation chapter reports a quick evaluation of the negation words handling in version two across the three datasets.

The three scenarios are described with an example each below.

1. A child of the current opinion node with a relationship of “neg”.

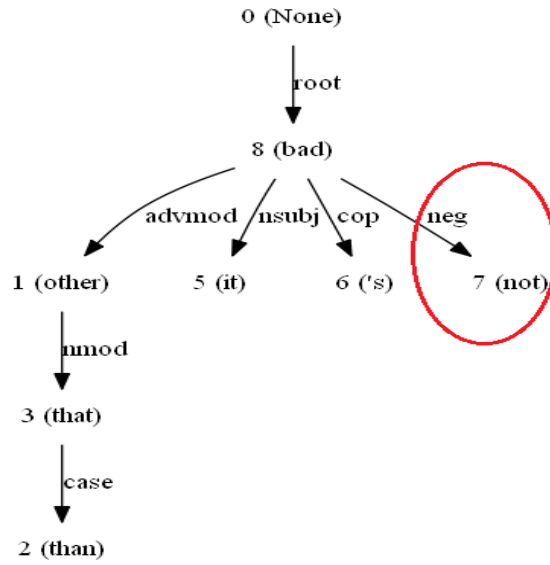


Figure 15 Negation word is a child of the opinion

2. A sibling of the current opinion node with a relationship of “neg” with the parent.

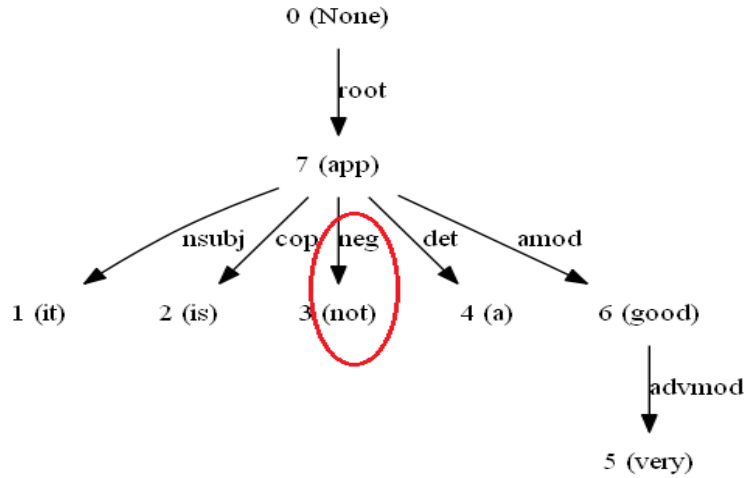


Figure 16 Negation word is a sibling of the opinion

3. The parent of the current opinion node with a relationship of “dep”.

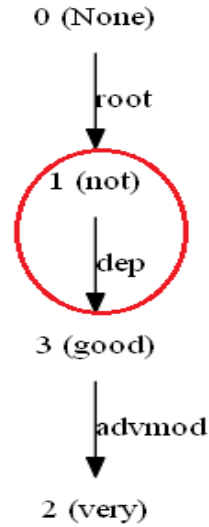


Figure 17 Negation word is the parent of the opinion

3.5.4 The algorithm in version one of the traversal function

Both versions of the traversal function take the same input and contain a list of pairing linguistic rules gauging topic-opinion pairs, which are introduced in the previous subsection 3.5.2 and in the Appendix D Topic-Opinion pair extraction linguistic rules. Both versions produce the same forms of outputs.

Version one is a simpler form of the traversal function and is suitable for the starting point to gather popular data, namely find out what the most popular topics or opinions are. The first part of it is finding the data of relation, child tag, parent tag and negation word, which form the relationship between the current node and its parent. The code gets the relation and the child tag from the current node first. Secondly, the code needs the parent tag. The parent can be traced up through the value of “head” of the current node. In order to locate the parent node from the node list to find its POS tag, it is necessary to swap the positions between the current node and its parent. The swapping is accomplished with the aid of a media variable “originalNode”. The positions are swapped back when the relationship between the current node and its parent has been checked against all pairing rules. This is necessary before the recursion is called.

The negation is considered in version one if a sibling of the current node has a “neg” relation with its parent. Negation words are handled better in version two. When a negation word is found, its address is taken as one of the four parameters that a related_pairs element holds, returned with it together if passing a rule’s checking, then eventually populated into the ontology as a part of the opinion.

When the whole relationship is checked against the pairing rules, the values of the relation, the child tag, and the parent tag are checked first. If they match a rule, the last field of the rule, which represents who is the “subject”, is queried. If the parent is the subject, the whole relationship is appended to the related_pairs in the form of parent address, child address, negation node address, and the rule number. If the child is the subject in the rule, the relationship is appended to the related_pairs in the form of child address, parent address, negation node address, and the rule number. Thereby, when the related_pairs list is finally returned from this traversal function, the sequence of each record is always the subject node address, opinion node address, negation node address, and the rule number. After the related_pairs returns, the addresses of each record are queried against the node list as indexes and the texts of the nodes are retrieved from the node list at the positions of the addresses.

Algorithm 3 The traversal function version one

input : An empty related pair list; a tree; the root node of the tree

output : The populated related pair list

parameter: A list of rules gauging Topic-Opinion pairs

Start this traversal function:

originalNode = node

if node has parent **then**

 Find relation and child tag;

 Swap position with its parent;

 Find parent tag and negation

end if

foreach rule in the list of pairing rules **do**

```

if relation, child tag, parent tag match a rule then
  if subject is "parent" in the rule then
    add this pair to the related pair list with the parent as the subject ;
  else subject is "child" in the rule then
    add this pair to the related pair list with the child as the subject
  end if
end if
end foreach
node = originalNode
if node has children then
  foreach child of the node do
    call this traversal function;
  end foreach
end if

After this traversal function:
Topics are compared with the "keywords" against ontology concepts.
Unwanted opinions are filtered by stopOpinions list.

```

Version one will return all data matching the rules, which is useful for stakeholders to gather the popular data for both topics and opinions. The data is stored in the database in the pairsTable under the fields of “subject” and “opinion”. A SQL query, such as:

```
select opinion from pairstable group by opinion order by count(opinion) desc;
```

or

```
select subject from pairstable group by subject order by count(subject) desc;
```

will produce the data in a sequence of frequencies. When applying this framework to new topic domains, this is helpful to understand what users are talking about most, therefore enables the refinement of the popular opinions list that will be introduced in next subsection 3.5.5 and the keywords array that will be introduced in subsection 3.5.6.

3.5.5 The algorithm in version two of the traversal function

Version two of this traversal function is an optimised version based on version one. It utilizes a popular opinions list to rule out unwanted opinion values, such as strange words or characters.

In the experiment, the popular opinions list is produced from both dataset one and dataset two, and leaves dataset three as the control dataset. 3000 highest frequent opinions are taken from dataset one. Another 3000 highest frequent opinions that do not appear in dataset one are taken from dataset two. The total 6000 non-duplicated highest frequent opinions are sorted in alphabetical order if they are adjectives, adverbs, nouns and verbs. Then a manual hand-pick process is performed with the author's judgement of whether each record is an opinion, with the help of a dictionary. Some miss-spellings are also accommodated into the popular opinion list.

A piece of code that helps this popular opinions mining process is provided in the list of code C.17. Users will take the highest frequent opinions from the database and convert them into a list of "opinions" in the code. This piece of code will then compare the list of "opinions" with the existing popular opinions list, and produce a sorted new opinion candidates list for the users to judge. Both the existing popular opinions list and the new opinion candidates list are in alphabetical order, thus it is convenient for the user in the process of updating the popular opinions list.

The resulting popular opinion list contains 1587 opinions, which is listed in the Appendix E Popular Opinions extracted from dataset one and two. Future researchers can take this opinion list as a starting point in the experiment and enrich it further with more datasets.

WordNet is not designed as a part of popular opinion tests in version two's algorithm on purpose, in order to rule out opinion word variances. For example, "balanced" is an opinion, but "balance" might not strictly be an opinion.

Different from version one, orphan opinions are picked up as much as possible in this version. Version two also has enhanced negation words treatment which is introduced in the previous subsection 3.5.3 Dealing with negation words.

Moreover, the logic of version two is also adapted to accommodate the enhanced negation words tests, and the popular opinion list. The logic is depicted in the Algorithm 4 Simplified pseudocode of the traversal function version two.

Such adaptations are performing two levels relationship checks (the relationship between the current node and its parent, and the relationships between the current node and its children), setting a flag whether the current node is an opinion, and filtering out duplicate records in the returned data in the later part of code where this version of function is called. Orphan opinions are implemented as a separate rule of pairing the orphan opinions with the app itself.

The purpose of carrying out two levels of relationship checks is to accommodate the orphan opinions search. The one level relationship check between the current node and its parent in version one is not sufficient for the decision whether a node is an orphan opinion. The scenarios that the current node is an opinion that forms relationships with its children are not checked in version one. Because version one does not carry out the orphan opinion tests, the single level relationship check is sufficient for the tree traversal. Whereas in version two, in order to decide whether an opinion is an orphan opinion, two level relationship checks are necessary. This can be illustrated in such two examples:

“My readings have never been better !” - (1)

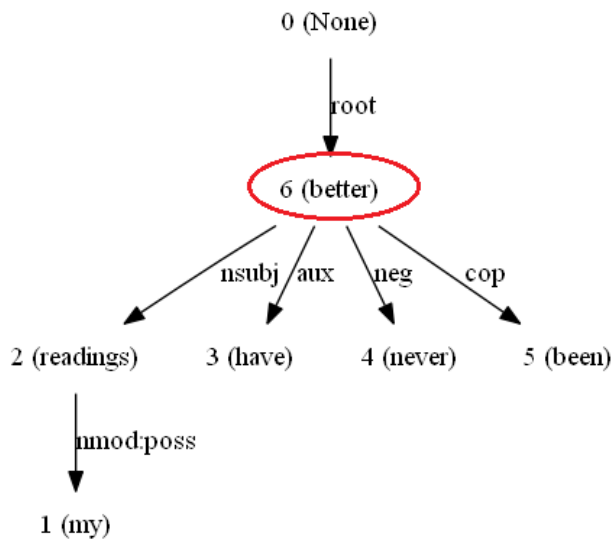
and

“Easy to use and has all the features that are needed .” – (2)

Sentence (1) treats root node “better” as an orphan opinion, whereas sentence (2) treats root node “easy” as part of a pair. These two trees are depicted in Figure 18 and Figure 19.

These two example sentences show that an opinion can only be certain to be an orphan opinion after the relationship between it and its parent and the relationships between it and its children both are checked.

A major benefit of such a two level relationship checking is that the current node’s position, whether it is a leaf or in the middle of the tree, is not essential anymore.

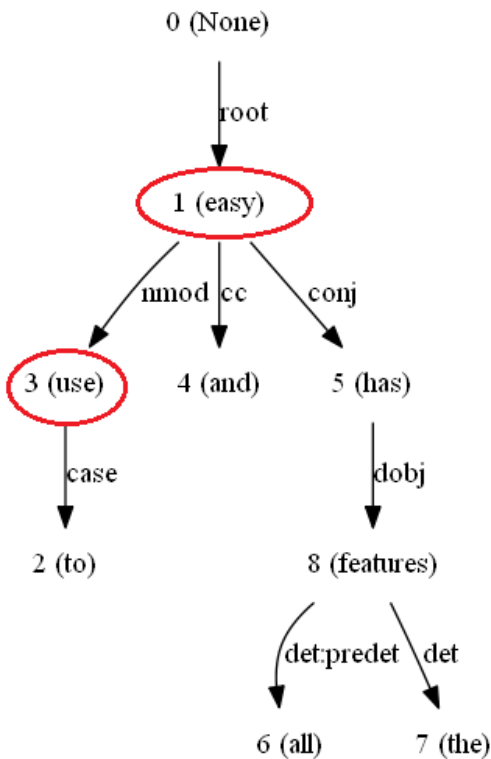


Root node “better” is an orphan opinion because it has no pairing relationship with its parent and any of its children. So the subject is assigned to “app”, which is a keyword associated with the “App” category in the ontology.

The output is as below:

Matched after keywords comparison:
 subject = app, negation = never, opinion = better, Rule Number is 1000, subjectCategory = App

Figure 18 Example sentence containing a root orphan opinion



Root node “easy” looks like an orphan opinion if only the relationship between it and its parent is checked. However, the two level relationship check also checks the relationships of it with all of its children. Therefore, one of its children nodes “use” is found to have a pairing relationship with “easy” that matches rule 110.

The output is as below:

Matched after keywords comparison:
 subject == use, negation == , opinion == easy, Rule Number is 110, subjectCategory == App

Figure 19 Example sentence containing a root opinion in a pair

Algorithm 4 Simplified pseudocode of the traversal function version two

```
input : An empty related pair list; a tree; the root node of the tree
output : The populated related pair list
parameter: A list of rules gauging Topic-Opinion pairs
The code below is under a condition that the current node is a popular opinion

Begin this traversal function:
originalNode = node
popularOpinionFlag = False
if node in popularOpinions then
    popularOpinionFlag = True
end if
if node has parent then
    Find relation and child tag;
    Swap position with its parent;
    Find parent tag and negation
end if
ruleMatchedFlag = False
foreach rule in the list of pairing rules do
    if relation, child tag, parent tag match a rule then
        if subject is "parent" in the rule and popularOpinionFlag = True then
            add this pair to the related pair list with the parent as the subject
            ruleMatchedFlag = True
        else if subject is "child" in the rule and popularOpinionFlag = True then
            add this pair to the related pair list with the child as the subject
            ruleMatchedFlag = True
        end if
    end if
end if
```

```

end foreach
node = originalNode
if node has children then
  try
    for each child of the node do
      Find negation
    end for
    for each child of the node do
      Swap position with its child;
      Find child tag and relation
      for each rule in the list of pairing rules do
        if relation, child tag, parent tag match a rule then
          if subject is "parent" in the rule and popularOpinionFlag = True then
            add this pair to the related pair list with the parent as the subject
            ruleMatchedFlag = True
          else if subject is "child" in the rule and popularOpinionFlag = True then
            add this pair to the related pair list with the child as the subject
            ruleMatchedFlag = True
          end if
        end if
      end for
    end for
    Swap position back with its child
  end for
except
  print ("there might be an index error")
end try
end if
if ruleMatchedFlag == False and popularOpinionFlag = True then
  associate this opinion with the app itself

```

```
end  
if node has children then  
    foreach child of the node do  
        call this traversal function  
    end if  
  
After this traversal function:  
Topics are compared with the "keywords" against ontology concepts.
```

Algorithm 4 shows an algorithm illustration of the version two of this traversal function that has been simplified.

As depicted in Algorithm 4, such checks are firstly performed between the current node and its parent, then between the current node and its children. The detailed process of each check is similar to version one as introduced in the previous subsection 3.5.4 The algorithm in version one of the traversal function. There are slight differences. Two flags are set up: flag for the current nodes being popular opinions, and flag for a linguistic rule has been matched. Moreover, when comparing the relationships with the linguistic rules, only the branches that have the current nodes as the opinions should be activated in the If statements. There is a minor bug at present in the code C.10, which will be corrected after the submission. Its impact on performance is trivial and has been discussed in a later section 5.5. After that, Algorithm 4 will also be updated.

Setting a flag about whether the current node is a popular opinion is not only for the code efficiency, but also for ruling out the complication of the two level relationship checks. The reason for setting up a flag for a popular opinion, instead of an If statement test before the two level relationship check, is that an If statement here will only allow popular opinions enter the check, and deny all the other nodes on the dependency trees. This is not appropriate because the tree traversal will not be completed, therefore errors are thrown later when a node that is not a popular opinion is encountered.

A flag about whether the current node is a popular opinion is set at the beginning of this traversal function. Each time when a relationship matches a pairing linguistic rule, it will

only be appended into the related pairs list when the popular opinion flag is true. In this way, the related pairs list only collects the relationships that relate to a popular opinion, and the tree traversal is not interfered by the popular opinion test.

Negation tests are enhanced because the negation words could appear as the parent, a sibling, and a child of the current node.

Removing duplicate records in the returned data when the function is called is achieved by this line of code:

```
related_pairs = list(dict.fromkeys(related_pairs))
```

Another flag about whether a linguistic rule has been matched by a relationship of the current node is also used in this version two's algorithm. The purpose of this flag is to separate normal linguistic rules check and the orphan opinions check. The initial value of this flag is False, but it will be set as True whenever a relationship matches a pairing linguistic rule and is appended to the related pairs list. At the end of the two-level relationship checking, if this flag is still False and the current node is a popular opinion, it is undoubtedly an orphan opinion.

When there is an orphan opinion, it is associated with the root node of the tree that has an address of "0", then the pair is appended to the related pairs list as usual. When the returned results are interpreted by the main code, it is easy to spot which node is a root node because a root node's text value is simply a "None". For such nodes that are in the position of the subject in a relationship, a value of "app" is assigned to the subject. By doing so, the orphan opinion is eventually associated with the app itself. For example, a resulting relationship can be: "subject = app, negation = "not", opinion = useful, Rule Number is 1000, subjectCategory = App".

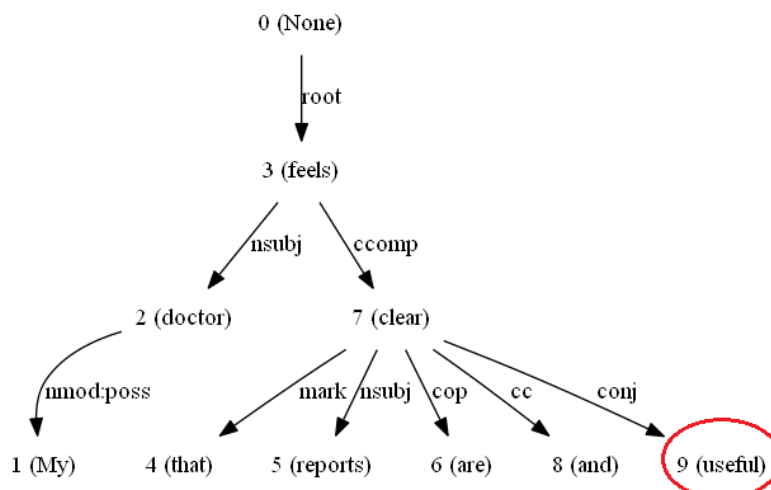


Figure 20 An example of orphan opinion

Orphan opinions are considered as opinions that have no topics associated. These opinions could be real orphans when there are no subjects for them in the sentences, such as a single word sentence “great”. They could also be left as orphans if the Stanford parser parses them separately away from their subjects. For example, in the sentence in Figure 20 “My doctor feels that reports are clear and useful”, “useful” is an opinion but it is parsed separately to its subject. Although it is linked with “clear” with a relation of “conj”, conjunction is not a relation being monitored in the pairing linguistic rules. The reason for not monitoring conjunction in this prototype is that conjunction does not normally relate topics and opinions directly. Therefore, there exists mutually exclusive interference between orphan opinions and the pairing linguistic rules design.

The pairing linguistic rules are targeting topic-opinion pairs as comprehensive as possible. The mutually exclusive interference between them and the orphan opinions becomes influential when a pair of relationships is caught by one of the rules but could not pass either the keyword filtering or the popular opinion test. Such examples exist frequently in short sentences with no subjects. For example, “very good” or “very very good”. The opinion “good” has no subject associated with it, so it should be caught as an orphan opinion. But it has adverb modifiers “very” associated with the relation of “advmod” as illustrated in Figure 21. The pairing rules will pick up such relationships, nevertheless “very” would not pass the keyword tests, and consequently such relationships will lose themselves.

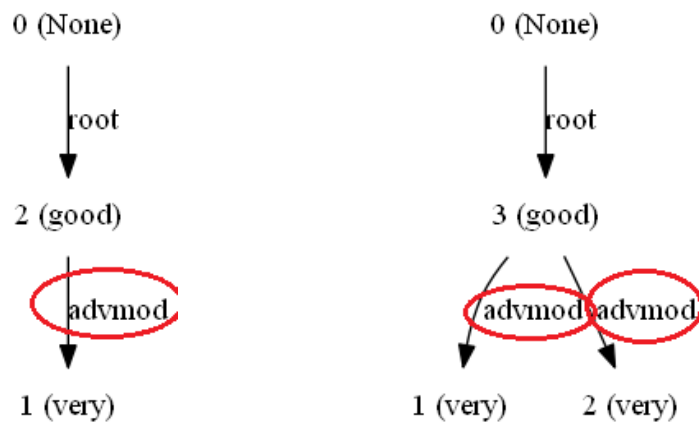


Figure 21 Examples of interference between orphan opinions and pairing rules

In an experiment illustrated in Figure 22, it is found that such phenomena exist frequently in two-word sentences. Among the 180 two-word sentences, 168 sentences should be picked up by the code ideally. However, only 112 sentences (62%) are picked up by the code. Among the 68 sentences that are not picked up, 56 sentences match certain pairing linguistic rules but lose themselves in the later keyword tests. Therefore, it is observed that there is an obvious drop in the pick-up rate from the one-word sentences to the two-word sentences, which could be about 30% more otherwise (93% instead in this case).

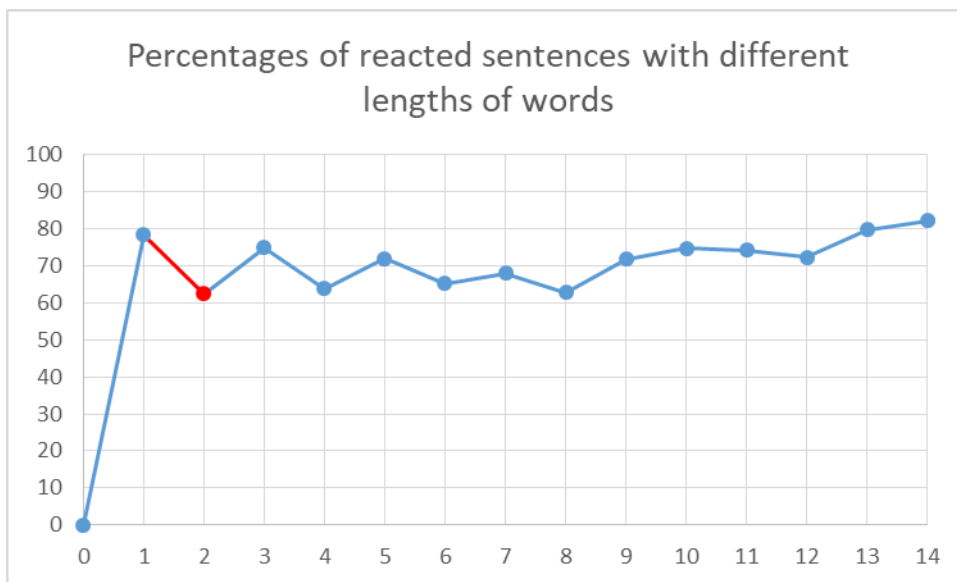


Figure 22 Observed decline in reacted rate for two-word sentences

3.5.6 Topic-Opinion pair filtering via keywords

This subsection introduces the second step of the Topic-Opinion Extractor component. As reflected in the example of how this component works in the first subsection 3.5.1 of this chapter, this step filters the previously produced arcs from Figure 13 to produce the final arcs in Figure 14. In other words, the task of this step is to distinguish the arcs and filter out the unwanted arcs, thereby producing the final arcs that map one or more classes in the ontology.

The topic-opinion pairs produced from the last step are stored in a database table. During this step, both the topics and the opinions go through a filtering process.

In order to help understand the general algorithm of this component, readers can imagine the whole approach as making “Sandwiches”. Firstly, prepare the materials by traversing the dependency trees with linguistic rules. The candidates of the relationship arcs are the materials to make these “Sandwiches”. However, in order to become ready for “Sandwiches”, both the two ends, topics and opinions, have to undertake certain filtering processes, just like to wrap the materials with two bread layers.

The filtering process for opinions:

Because there are two different versions of the traversal function in the previous step, the opinions filtering is also different in this step accordingly, which is reflected in two versions.

Corresponding to version one of the traversal function, the filtering of opinions is achieved through a list of “stopOpinions” in C.11. During this process, some unwanted opinions are filtered out by “stopOpinions”. The disadvantage of this version is that the list of “stopOpinions” is manually maintained. However, if this list of “stopOpinions” is maintained well, the strange strings that flee from the opinion filtering are normally low in frequencies. Therefore, the resulting strange opinions don’t affect the performance much. They just affect the data looking in the ontology. The list of code C.11 contains the opinion filtering corresponding to the version one of the traversing function in C.9.

Corresponding to version two of the traversal function, the filtering of opinions is achieved through the list of “popularOpinions”, which has been implemented in the code C.10. How the popular opinions are gathered has been introduced in the previous subsection 3.5.5. The

list of code C.12 does not enable the list of “stopOpinions” filtering. C.12 is corresponding to version two of the traversing function in C.10.

Both C.11 and C.12 contain the same filtering process for topics.

The filtering process for topics:

The topics are compared with a two-dimensional array, which contains keywords corresponding to classes under the “Subject” class in the ontology. This is a classification process in this framework that maps the topics onto the ontology. The literature of this concept mapping onto ontology is briefly explored in section 2.3 in the Literature review chapter. The detailed method on how this is accomplished is reported in this subsection.

It is a debatable topic about what the most formal way to classify and map the topics onto the concepts in the ontology is. There are plenty of varieties in the literature on this question that apply different methods. In this component, the classification and concept mapping are realized in a very simple and light-weight method via a two-dimensional array. This is an alternative method to the formal ways of classification and concept mapping and is suitable in this specific framework.

There are three questions below in this classification process.

Q1. How to know which user reviews are talking about a concept in the ontology?

Q2. What if users use different but similar words with the terminologies?

Q3. How to actually map the pairs onto the concept categories in the ontology?

Question Q1 is resolved through a list of keywords for each concept, which forms the first dimension of the array. The two-dimensional array consists of such an array for each concept in the ontology. In the list of code C.11 and C.12, the topic of each topic-opinion pair is compared with each keyword in the array.

These keywords are carefully decided by manual selection initially, and enhanced by the high frequency keyword analysis during the data evaluation process. An example of 'UserInterface' below explains how to design and enhance this array in the future.

Table 23 An example of 'UserInterface' in the two-dimensional keywords array

```
['UserInterface', 'background', 'button', 'buttons', 'colour', 'color', 'colors', 'dark', 'display', 'emojis', 'fields', 'font', 'giggling', 'green', 'gui', 'icon', 'icons', 'interface', 'interfaces', 'layout', 'user interface', 'light', 'message', 'messages', 'noise', 'noises', 'notification', 'notifications', 'popup', 'popups', 'presentation', 'red', 'screen', 'screens', 'search', 'settings', 'sound', 'sounds', 'size', 'scroll', 'scrollbar', 'toast', 'ui', 'view', 'voice'],
```

All the keywords in the lower cases are the keywords to look for from the user reviews. They are decided gradually from the manual data analysis initially. Furthermore, 'voice' is included from the data evaluation process. During the evaluation against dataset one and two, 'voice' appears in the top 500 frequent words. Although in dataset one, 'voice' appears only 3 times, it appears in dataset two 206 times, thereby positions at the top 201st most frequent word in dataset two. The SQL query that statisticizes the subject words by frequencies in the previous subsection 3.5.4 helps the decision making in the keywords enhancement.

Readers might wonder why 'toast' appears in this keyword list of 'UserInterface'. This 'toast' is a term for mobile app developers to design a temporary popup message that disappears in a few seconds. But in practice, it is frequently linked to “bread” with the involvement of WordNet. Future researchers are recommended to make their own decision on whether to keep this term.

To resolve the question (Q2) of recognizing similar words being used in the reviews, WordNet is used in this component.

If a topic term is the same word with any keyword in the array, this specific topic-opinion pair is classified into this category and recorded into the database in the form of the pair id and the category, which is the class name in the ontology. The pair won't be classified into duplicated categories because duplications are filtered out in the classification.

If the topic term is different with all the keywords in the array, it is then compared with the keywords in the context of WordNet for similarities. A similarity is calculated between this term and the keywords. If the similarity is above the predefined threshold, and this specific topic-opinion pair has not been classified into the same category previously, it is regarded as

related to this category and recorded into the database in the same form of pair id and the category.

It is worth to mention that, above two comparisons are accomplished in a same loop that loops through the keyword array for each topic-opinion pair just once.

The reason to separate the above two comparisons is that some topic terms are not in WordNet. For such terms, WordNet throws errors because it cannot calculate similarities for terms that it does not know.

For the sequence of above two comparisons, it is better to put the equal comparison before the WordNet similarity calculation. This is a decision after a series of experiments that suggest WordNet performs worse for exact same word similarity calculations than for the different words. For example, the same word “enter” and “enter” have a similarity of 0.5 when 'wup_similarity' is used. Whereas 'wup_similarity' performs the best in the experiments compared with other similarity calculations.

The similarity threshold is set to 0.90 for 'wup_similarity' in this component in the deliverables. This threshold provides good performance for most cases, although it is still awkward to merge singular and plural forms. It is also arguable whether it is too generous in the case of the 'toast' above, in which WordNet thinks “bread” is very similar to the 'toast'. However, the experiments suggest that wup_similarity at 0.90 outperforms other algorithms.

The answer to the question Q3 is inside the design of the array and the above loop. This is a simplified solution that puts the category (the class) name at the first position in each of the single-dimensional arrays that represents one class in the ontology. All the single-dimensional arrays together build up the two-dimensional array. When looping the array for each topic-opinion pair and the pair is found related to a specific keyword (keywords[i][j]), the value of the first keyword in that line (keywords[i][0]), which is the name of the class in the ontology, is assigned to the variable “subjectCategory” for this pair. This “subjectCategory” is written into the database with the pair id together in order to remember which topic-opinion pair belongs to which class in the ontology.

For the full logic of this step and how above questions are resolved, readers are referred to the list of code C.11 Topic-Opinion pair filtering via keywords Version one and C.12 Topic-Opinion pair filtering via keywords Version two.

This keyword comparison step works together with the previous step Topic-Opinion pair extraction via linguistic rules to produce the pairs that can be mapped onto the ontology. In the previous example sentence used in subsection 3.5.1, the original topic-opinion pairs that match the linguistic rules are six pairs. But the final matching pairs are two. The Figure 23 below depicts the result that the debugging code C.13 produces for this example. The debugging code is introduced in a later subsection 3.5.8.

```
inserted according to rule 49: 8, 7 and -20
inserted according to rule 11: 6, 7 and -20
inserted according to rule 17: 13, 12 and -20
inserted according to rule 100: 13, 19 and -20
inserted according to rule 112: 15, 19 and -20
inserted according to rule 49: 18, 17 and -20
Matched after keywords comparison: subject == presentation, negation == , opinion == clear
, Rule Number is 49, subjectCategory == UserInterface
Matched after keywords comparison: subject == managing, negation == , opinion == easier,
Rule Number is 100, subjectCategory == Function
>>>
```

Figure 23 Example result of Topic-Opinion pair extraction

Lastly, there is a point that needs future researchers' attention in this piece of code. This code contains the WordNet snippet in the loop for database query result processing. WordNet looks consuming considerable memory, which causes the code unexpectedly quits in the experiments when there are printing statements in the code. The solution to the unexpected quit is removing all printing statements from the code, including the one in the "except" block. After this adaptation, the code C.11 and C.12 never quit unexpectedly.

3.5.7 Data flow connector between the Request Elicitor and the Topic-Opinion Extractor

The overall structure of this framework is batch sequential style Data Flow architecture, which is introduced in Figure 1 in section 1.5 Top level structure and Figure 5 at the early part of the Methods chapter. It is self-evident that the next component takes input from the output of the previous component. However, it is necessary to explain the data flow connector between the Request Elicitor component and the Topic-Opinion Extractor component. The

speciality of this data flow connector is that the latter component takes input from the sentences that did not pass the former.

The reasons for this design are based on a few assumptions below:

- When users express a request in a sentence, it is assumed that the request has a higher priority for researchers and stakeholders than an opinion.
- It is also assumed that sentences that have user requests seldom contain opinions or it is acceptable to ignore the opinions in such sentences based on the above assumption.
- If opinions appear in a sentence that does not mention a new request, the opinions are related to existing aspects of the mobile apps, namely existing requirements.
- Researchers and stakeholders will be benefited from this convenient separation format when researching opinions against existing requirements and requests against new requirements.

Therefore, the resulting requirement in the data flow connector that integrates between these two components is that sentences that contain user requests do not go through topic-opinion extractor again.

There are several plans to implement this requirement. From the code structure aspect, a bigger piece of code could have been produced that would merge the two components together into one. From the data structure point of view, an extra data structure could have been created in the latter piece of code to tackle the problem of ruling out the data that have already been processed by the former component. Although these plans could solve the problem, they either make the code cumbersome, or have to use a certain amount of extra computation and memory.

Eventually the final solution applied Left Join between the tables that the latter component takes and that the former component produces, when the latter component takes input. By explanations in detail, both two components, the user request elicitor and the topic-opinion extractor, take input data from the same data source table, “tokenizedsentences” table. The user request elicitor takes the input data from this table first and produces user requests that are stored in “requestsTable”. When the latter component, the topic-opinion extractor, takes

input, it has to consider the data from both tables, “tokenizedsentences” table and “requestsTable”. It only processes the sentences that exist in the “tokenizedsentences” table and have not produced a request in the “requestsTable”.

The reason for a Left Join can help in this case is that Left Join produces an empty record with a NULL value for each row in the position of the field from the right table if this row of record exists in the left table but not in the right table. Utilizing this nature, a Left Join is made between the “tokenizedsentences” as the left table and the “requestsTable” as the right table. Then sentences, which exist in the left table but have no request record in the right table, are taken as the input data for the latter component, topic-opinion extractor. The SQL statement that fulfils this requirement is listed below:

Code Snippet 16 SQL query fulfils the data flow connector between request elicitor and topic-opinion extractor

```
SELECT * FROM tokenizedsentences t left join requestsTable r on t.reviewId = r.reviewId  
and t.sentenceId = r.sentenceId where r.request is NULL ;
```

This method successfully eliminates the data processing when the latter component takes input from the database. This is regarded as the easiest way to integrate the Request Elicitor and the Topic-Opinion Extractor that costs the least.

3.5.8 The debugging code to help future adjustments

There are two pieces of debugging code in the deliverables of this thesis that correspond to the two versions of the traversal function respectively:

C.13 Debugging code to help topic-opinion pair extraction and filtering Version one

C.14 Debugging code to help topic-opinion pair extraction and filtering Version two

Both pieces of code above merge the functionalities of the two steps of this component together, and give instant results of the input sentence, without interaction with the database. They aim to serve future researchers in their process of developing linguistic rules for this component.

The logic of the debugging code is the combination of step one (C.9 or C.10) and step two (C.11 or C.12), apart from no statements talking to the database. The “text” variable takes a sentence as the input. This code instantly produces the result that will be populated into the ontology with regard to this sentence. The result looks like the one in Figure 23 Example result of Topic-Opinion pair extraction.

For the visualized dependency tree structure, readers need to uncomment the four lines under the comment “##### the four lines below is a switch between the dependency tree graph and the code output”. The reason for these four lines being separated is that these four lines of code uses “graphviz” to draw the tree picture, which perhaps handles the problems of non-consecutive node indexes and multiple nodes appearing as one node in another way. If these two problems happen, these four lines of code will crash the debugging code. When they are separated out, the behaviour is generated by “graphviz”, and any error is also thrown by “graphviz”. Currently, an error is thrown, but it is from “graphviz” and it is after the tree picture is generated. Therefore, it is still helpful for future researchers to visualize the sentence’s dependency structure in a picture of a tree.

The suggested adjustments, which future researchers could do and at the same time they do not need to change the structure, are discussed below:

- Adjustments to the “keywords” array in order to control the concept mapping onto the ontology that will facilitate the adaptation to new topic domains and environments.
- Adjustments to the “rules”, in order to enrich them in new environments. Since the rules are in OCP principle, they could even be replaced by another set of rules completely for another research question.
- Adjustments to the “stopOpinions” in order to block unwanted opinions entering the ontology in the code of version one.
- Adjustments to the “popularOpinions” list in order to accommodate more opinions and adapt to new environments for the code of version two.

More adjustments that need structure changes are discussed in the 5.6 Future research directions.

3.5.9 Explicit opinions plus a little bit of implication for the “app” topic

In this implementation, similar to the Request Elicitor, this Topic-Opinion Extractor component also mainly works on explicit opinions. This is to say that implicit opinions are generally not extracted, such as there are no explicit opinion words in the sentences. When people tell stories in a narration way and do not mention any explicit opinion words, it is difficult for a computer program to mine the opinion behind. However, such sentences always exist in user reviews. This is reflected in the Results and evaluation chapter. When the reasons for the false positive and false negative results were analysed in section 4.4, “no obvious opinion term” constitutes one of the reasons across all samples from the three datasets.

However, one exception has been made towards the concept of mobile apps themselves. Two forms of implications towards to the current mobile apps are handled in this component:

- (1) Orphan opinions that have no subjects associated
- (2) Opinions that associate to subject terms implied to the current mobile apps

Orphan opinions are handled in version two of this component. The way of how they are handled is introduced in the previous subsection 3.5.5 when introducing the algorithm of version two.

The topics of the current mobile apps are frequently mentioned in user reviews and relate to opinions. When users express opinions to the ‘app’, mostly it means the current mobile apps. Users also express opinions to the current mobile apps in an implication way: mentioning topics that can be implied to the current mobile apps. This can be seen in the forms of 'content', 'thought', 'detailed', 'work', 'tool', 'designed', 'motivation', 'product', 'program' and some other words. Even the pronoun “it” means the same concept most of the time.

Therefore, both versions of this component deal with those “app” related terms implications. This is achieved through the keywords list. When the implied subject terms are mentioned, they are classified to the “App” category. The definition of the “App” category in the keywords list is below:

['App', 'app', 'application', 'content', 'database', 'design', 'designed', 'detailed', 'game', 'handles', 'help', 'helped', 'improvements', 'informative', 'it', 'job', 'motivation', 'motivator',

'product', 'program', 'rating', 'see', 'seeing', 'stuff', 'thing', 'thought', 'tool', 'use', 'way', 'work', 'works']

This decision is supported by the high frequencies of such terms in the user reviews. The high frequency trend remains the same in the results of all three datasets. The table below compares the frequencies of the term “app”, “app” related terms, and the rest of other terms among the three datasets in the “pairsTable”.

Table 24 Frequencies of "app" and related topics across three datasets

	Dataset 1	Dataset 2	Dataset 3
(1) Frequency of term “app”	8262	31452	7186
(2) Average frequency of “app” and its related terms	776.2333	2774.4483	1542.7857
(3) Average frequency of other terms	13.4702	20.6133	12.8182

Table 24 uses the results of version one code C.9 in this component to get above data. This is because version one code writes to the “pairsTable” without the test of popular opinions therefore is suitable for the initial subjects and opinions data gathering and analysis that help to produce better performance in version two.

The SQL statements that produce the data in Table 24 are listed below:

(1) Frequency of term “app”:

```
Select subject, count(*) from pairsTable where subject = “app” group by subject order by 2 desc;
```

(2) Average frequency of “app” and its related terms:

```
SELECT AVG(a.pcount) FROM (Select subject, count(*) as pcount from pairsTable p where subject in ('app', 'application', 'content', 'database', 'design', 'designed', 'detailed', 'game', 'handles', 'help', 'helped', 'improvements', 'informative', 'it', 'job', 'motivation', 'motivator', 'product', 'program', 'rating', 'see', 'seeing', 'stuff', 'thing', 'thought', 'tool', 'use', 'way', 'work', 'works') group by p.subject order by 2 desc) a;
```


(3) Average frequency of other terms:

```
SELECT AVG(a.pcount) FROM (Select subject, count(*) as pcount from pairsTable p where
subject not in ('app', 'application', 'content', 'database', 'design', 'designed', 'detailed',
'game', 'handles', 'help', 'helped', 'improvements', 'informative', 'it', 'job', 'motivation',
'motivator', 'product', 'program', 'rating', 'see', 'seeing', 'stuff', 'thing', 'thought', 'tool',
'use', 'way', 'work', 'works') group by p.subject order by 2 desc) a;
```

It can be seen from the comparisons in Table 24 that “app” and its related terms are mentioned much more frequently by users than other terms in the reviews. It may be also worth noting that the frequencies of the term “game” are 12, 328, and 14909 in the three datasets respectively.

Implicit opinions are harder to mine than explicit opinions in general. This is not only because that implicit opinions have more complicated rules or no rules, but also because that negation words are harder to handle. Some attempts have been made in the experimentation stage of this prototype for certain implicit opinion mining, but failed due to awkward handling of negations. Negation words are much easier to handle with the help of the Stanford dependency parser. That has been reported in the previous subsection 3.5.3 and the evaluation is reported in a later section 4.5 in the next chapter.

For future researchers who aim to deal with implicit opinions in user reviews, combinations with dependency trees would be a recommendation in this direction since Stanford dependency parser has relatively mature handling for negation words.

3.6 General Mobile App User Review Ontology (GMAURO)

This section introduces the last component of this framework, GMAURO. All the resulting data will be populated into GMAURO that provides facilities for users to query about the results in ontology. Screenshots in this section are taken from Protégé, a graphical tool to show users the data in ontology. Subsection 3.6.1 presents the GMAURO design. Subsection 3.6.2 proposes an algorithm to populate complex data from the database. Subsection 3.6.3 introduces methods to break a large population process down to small units. Subsection 3.6.4 reports the population code structure. Subsection 3.6.5 introduces a method to read from the existing data during the population in order to prevent them being washed out. Subsection 3.6.6 describes the way to merge negation words with opinion words in this prototype. Subsection 3.6.7 reports some measures that have been taken to prevent the ontology crashes. GMAURO is evaluated in section 4.6 of Results and evaluation chapter.

3.6.1 Ontology design

GMAURO is designed to accommodate requirements from stakeholders' viewpoints, which corresponds to viewpoint-oriented requirements. A "viewpoint" is an important requirement elicitation technique in Software Engineering and Requirement Engineering [110]–[112]. The generalized topics designed in the ontology GMAURO consider viewpoints from different stakeholders, such as users, requirement engineers, developers (including UI developers).

Figure 24 to the right is the class hierarchy of GMAURO. All classes under the "Subject" are subject hierarchy representing requirements from stakeholders' viewpoints. "Request" is the class to store new user requirements elicited from the automatic user request elicitation process. "Feature" and "Function" represent existing requirements that can be traced in the requirement engineers' viewpoint. Classes under the "Category" emphasize the developers' viewpoint, while "UserInterface" emphasizes more on the user interface developers' taxonomy. The "Price" and "Version" classes are attempts to elicit user feedback on price and version control, which may be optimized in more sophisticated ways.

“Attribute” is a class of attributes that can describe individuals under “Category” with more detailed information. However, this detailed information is not developed in this version of implementation. Future researchers are strongly recommended to improve this ontology and make use of the “Attribute” class.

“Opinion” is a class that hosts all the individuals for opinions. All individuals in “Opinion” class have two object properties: “isOpinionFromReview” linking to “some” “Review” and “opinionIsRegardingSubject” linking to “some” “Subject”.

“Review” is a class that hosts all the individuals for reviews. The individual reviews all have two object properties: “reviewHasOpinion” linking to “some” “Opinion” and “reviewIsTalkingAboutSubject” linking to “some” “Subject”. The “reviewHasOpinion” property is an inverse property for “isOpinionFromReview” and vice versa.

“Subject” is a superclass for all types of topics that a review could talk about currently. There are subclasses “App”, “Category”, “Device”, “Feature”, “Function”, “Price”, “Version” and “Request” so far. These classes are not disjointed in this ontology. Similarly, the first level of four super-classes are not disjointed as well. This design is to accommodate concepts that appear as more than one classes in the same level.

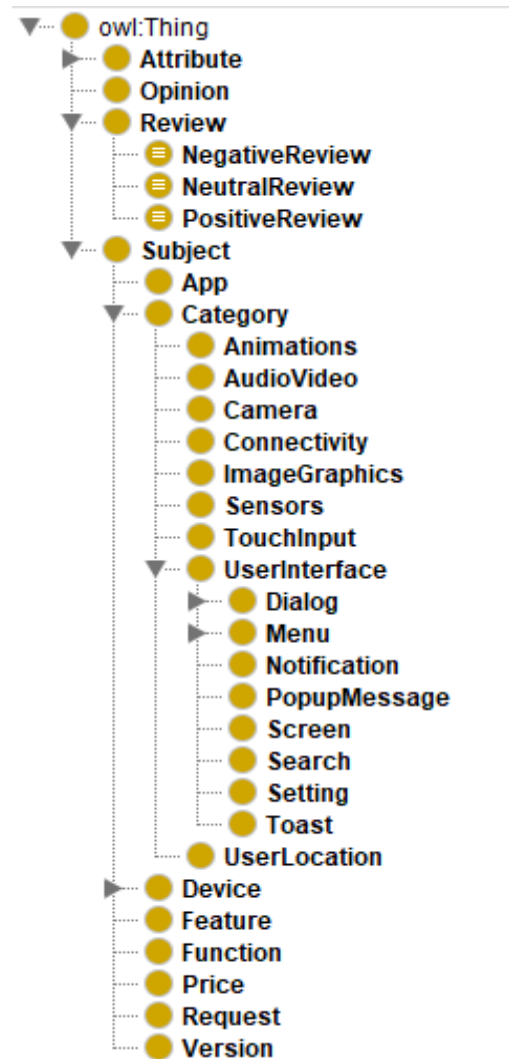


Figure 24 GMAURO class hierarchy.

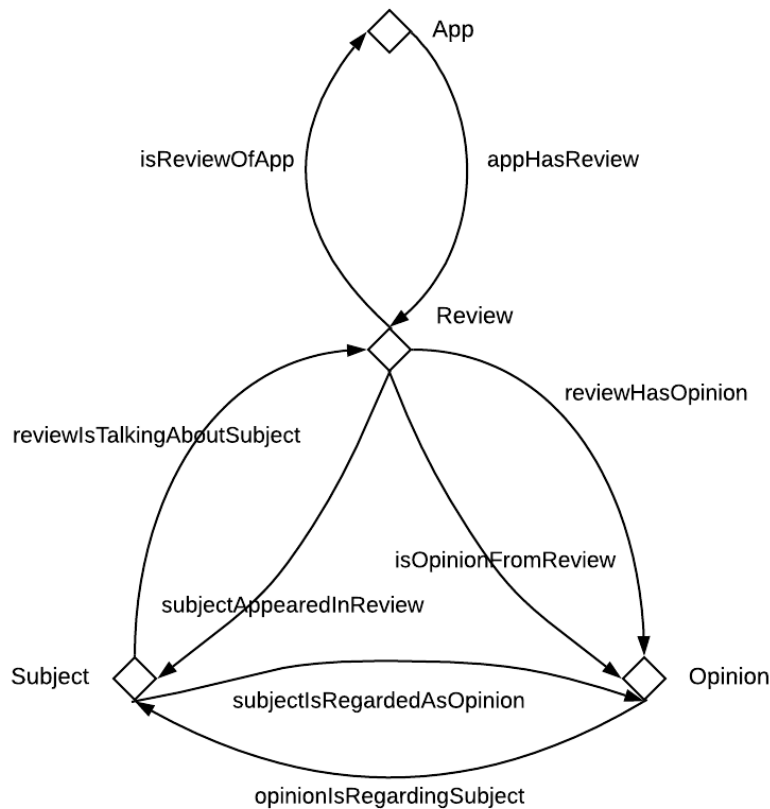


Figure 25 The inverse relationships among the six object properties

All individuals under “Subject” have at least two object properties: “subjectAppearedInReview” linking to “some” “Review” and “subjectIsRegardedAsOpinion” linking to “some” “Opinion”. The “subjectAppearedInReview” is an inverse property for “reviewIsTalkingAboutSubject” and vice versa. Similarly, the “subjectIsRegardedAsOpinion” property is an inverse property for the “opinionIsRegardingSubject” property and vice versa.

It is important to note that none of those above object properties is functional because the relationships all are many to many. Similarly, none of them are inverse functional properties.

Moreover, “App” and “Review” also have a pair of inverse object properties: “appHasReview” linking to “some” “Review” and “isReviewOfApp” linking to “exactly” 1 “App”.

The mutual inverse relationships among those above eight object properties are depicted in Figure 25.

The mutual inverse relationships among the eight object properties attempt to maintain the traceability among App, Review, Subject, and Opinion classes.

The “Review” class also has three sub-classes “NegativeReview”, “NeutralReview”, and “PositiveReview”, which are disjointed one another. All these three classes are equivalent classes that enable reasoners automatically infer the sentiment type of each review. The definitions of these three classes are below:

Table 25 Equivalent classes of three types of reviews

Negative review:	Review and (hasSentimentScore exactly 1 xsd:double[< "0.0"^^xsd:double])
Neutral review:	Review and (hasSentimentScore exactly 1 xsd:double) and (hasSentimentScore value "0.0"^^xsd:double)
Positive review:	Review and (hasSentimentScore exactly 1 xsd:double[> "0.0"^^xsd:double])

When a reasoner is turned on, reviews are automatically inferred as one type of review according to their sentiment values. The inferred types are highlighted as a light yellow colour as in the figure below.

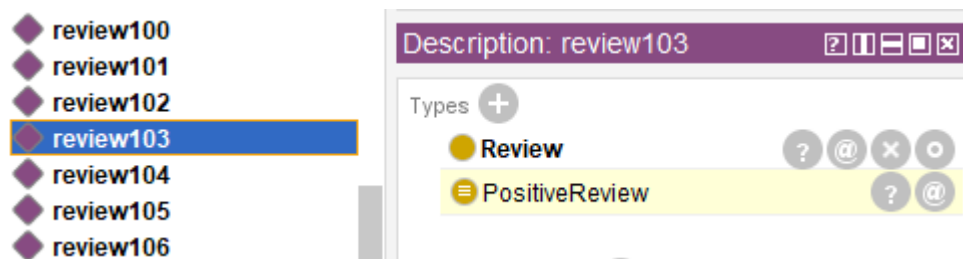


Figure 26 A reasoner highlights the inferred type of a review

There are also a number of data properties in this ontology. Both types of properties are presented in Figure 27. The highlighted “keywords” property in Figure 27 is optional for the concept mapping from the topics mined from the previous component to this ontology. The

purpose of this property is for researchers who prefer to use conventional methods to compare the previous keywords array with some data source in the ontology itself. Since a simpler approach of a two-dimensional array has been taken in this version of implementation, the “keywords” property is not used currently.

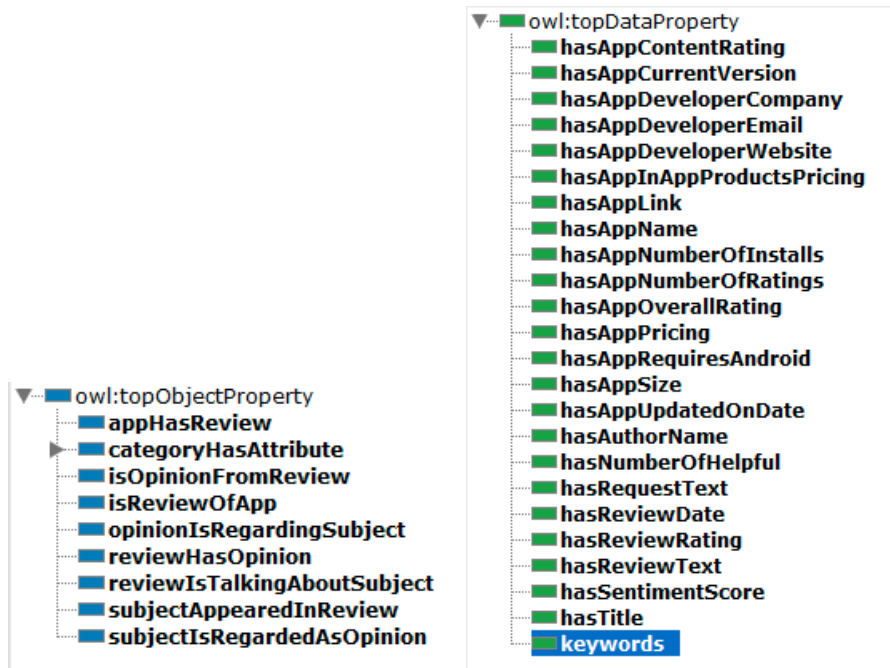


Figure 27 Object properties and data properties in GMAURO.

Because of such design, users can achieve easy navigation in the concepts of this GMAURO ontology. For example, from app to reviews, from a review to the opinions and topics that appear in this review, the descriptive relationship between opinions and topics, and vice versa. Figure 28 shows the easy navigation from an app to the reviews and the properties of the app in the left picture.

Moreover, easy navigation is also provided by Protégé itself. Individuals are grouped by classes and accessible via the “Individuals by type” tab. This tab is convenient to show what individuals are for each type, and how many they are. Figure 28 also shows the “Individuals by type” tab after the population in the right-hand side of the picture.

A series of SPARQL queries are delivered with the framework together to answer some common questions. For the detailed SPARQL queries and the questions they answer, readers

are referred to the 4.6 section in Results and evaluation chapter. Users are also recommended to inspire their own SPARQL queries to answer more questions.

Snap SPARQL Query is recommended to use for the SPARQL queries. The reason to recommend it is that Snap SPARQL Query is very fast in opening and actions. As a contrast, the Protégé built-in SPARQL Query tab takes unreasonable long time to open for a big populated ontology. For example, for the resulting ontology from the dataset one, the built-in SPARQL Query tab takes 5.5 hours to open the tab on the author’s computer, provided the memory setting “-Xmx” of Protégé run.bat has been increased from 500MB to 30GB.

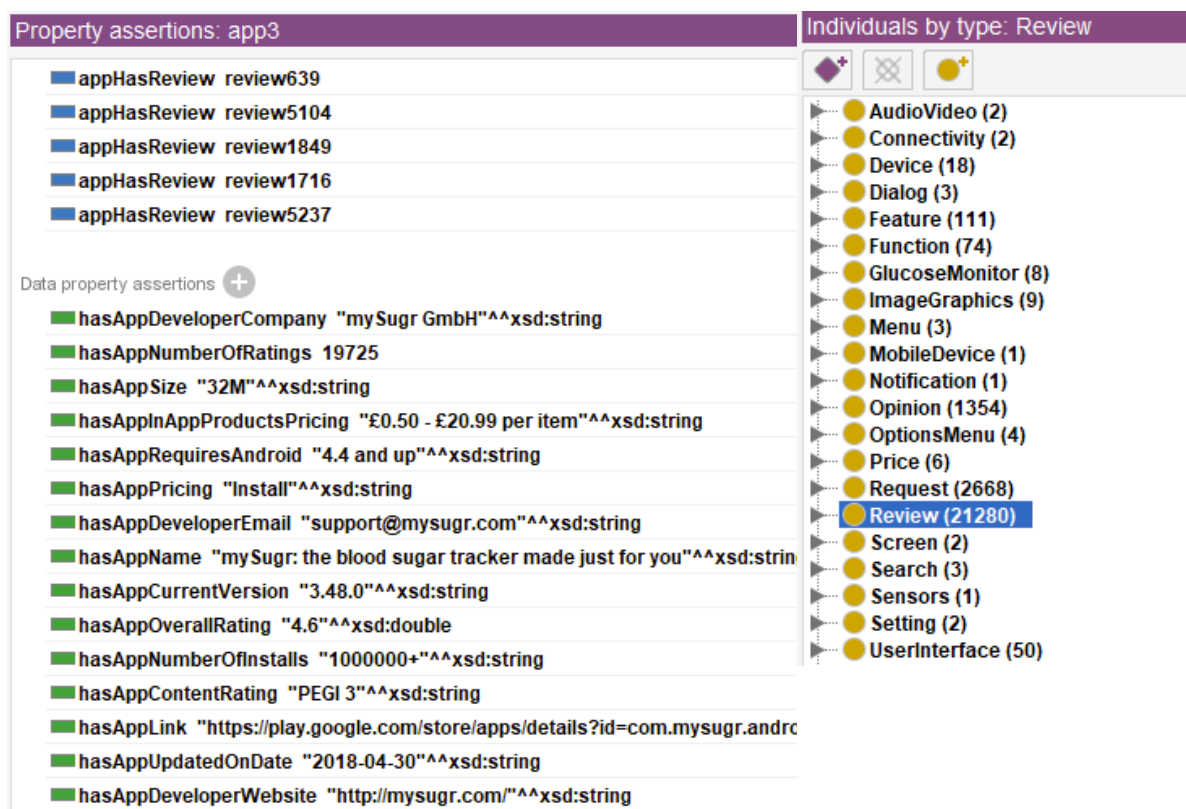


Figure 28 Easy navigations in the ontology

One more benefit of Snap SPARQL Query is that it supports querying over inferred knowledge. Though this is not used in the evaluation due to the large size of the ontologies. The reasoners take an unacceptable long time to start for the ontologies that are populated with the datasets.

In order to use Snap SPARQL Query, it is needed to create a new tab. The Snap Plugin is available as a View: Window > Views > Query Views > Snap SPARQL Query. Then drop it in the tab that is just created.

Technically there are more possibilities to proceed further to visualize the answers if piping the data from the answers to a data visualization software, such as a spreadsheet software or a statistical tool. Different SPARQL queries produce different answers that vary in the number of fields. Future researchers are welcome to handle this process in different ways. This is not included in this implementation of the deliverables.

3.6.2 An algorithm for complex ontology population

All the data of individuals, relationships, and data properties are populated into GMAURO through an ontology populator. The populator thoroughly populates all above data from the database. This actually ensures a complete web being built upon the whole available reviews and enables stakeholders to do analysis on a full picture of the apps with no background data missing.

In this framework, the process of data population into the ontology is designed in a Java project that has been optimized after experiments. For the source code of this Java project, readers are referred to the list of code C.15 The ontology populator source code (a Java project).

When data complexity is low, the population process can be easily managed through one cycle of these steps: requesting data from the database, building objects, setting up relationships, assigning property values, and populating them into the ontology. When multiple tables involve, it is necessary to join tables at the beginning step: requesting data from the database.

However, along with the increasing data complexity, one cycle of those steps could be insufficient. When multiple such cycles happen with different data targets, two problems of data missing will emerge if they have not been foreseen and tackled.

The first one is a phenomenon that new data will wash out the previous data if they are populated again in another cycle. The ontology can only keep the property values and relationships from the newest population cycle for the same data. This will cause the previous

data, which is expected to exist, to go missing. The solution for this problem is to read the existing data from the ontology and update them with extra information, instead of populating them again during the latter cycle.

The second problem may still exist: some data may still appear as missing after the first problem is fixed. These could be missing links between data if they are not set up among different cycles. Although increasing the number of joining tables to build comprehensive data links during each cycle could be a direction, it can become very cumbersome, yet sometimes not possible.

In addition to the above two problems, background data are not populated as well. This is normally not an obvious issue in a single cycle of population if only the intersection data of the joining tables are needed. However, if the derived data analysis is based on the whole dataset corresponding to the database, the set differences between the resulting intersection data from one cycle of population and the whole dataset will matter.

In order to overcome all above three problems, a simple algorithm is proposed. This algorithm is different to the typical single cycle of ontology population code where multiple cycles of populations can be avoided. The main method of this algorithm is to populate all the candidate database tables in turn. Candidate database tables are the tables containing the data that are required by the ontology. In other words, there still can be some tables that are not populated. Additionally, joining tables is only avoided to the greatest extent, but not excluded completely. It is still used if it is inevitable. The key of this method is a list of tables or joining tables that can be populated in turn as separated as possible without creating troubles. The first table should be a base table of the database that has no foreign key referencing other tables. Then the second table could reference the first one. Other tables follow hereafter on the list in turn on the same principle. The logic of the algorithm is presented in Algorithm 5.

This algorithm's logic is behind the ontology populator coding implementation, and not necessarily convertible into explicit code.

This ontology population algorithm provides a possibility for researchers to populate complex ontology, even from the whole database or databases. If the source database(s) is big, the resulting ontology will be big as well, which will require longer population time.

Algorithm 5 Algorithm for complex ontology population

```
Input: A list of tables or joining tables that can be populated in turn as separated as possible
Output: An populated ontology that might have complex structure

while the list is not empty do
    Take one item from the top of the list;
    if there are existing data in the ontology that are the same individuals with this item
then
        Read them from the ontology;
        update them with the new information in this item;
    else
        Populate data from this item into the ontology;
    end if
end while
```

3.6.3 Methods to break down large ontology populations

When populating a very large ontology, if the computer is not a supercomputer, the process will become very slow in the later population process. If any error is spotted, the whole semi-populated work is wasted.

Code Snippet 17 The method to execute the population algorithm item by item

```
public static void main(String args[]) {
    /**
        // Code to populate Apps.
    **/
}
```

```

    /// This is the code that is currently executed!
    // Code to populate Reviews.
/**
    // Code to populate Requests.
    // Code to populate Topic-Opinion pairs.
**/
}

```

One method to avoid these problems and make the population more manageable is to break down the population process to the level of table by table. Namely, having the list of tables or joining tables from the above algorithm, execute this algorithm item by item from the list. This can be achieved through commenting out other items' population code. Code Snippet 17 depicts this method with an example.

Another line of thought on the methods avoiding the problems when populating a large ontology on an ordinary computer suggests that researchers can even break down one item's population further by manipulating the size of records being populated each time, provided if any one item is large enough and has to be broken down into smaller units. This method is illustrated in the example in Code Snippet 18.

Code Snippet 18 The method to break down records of one item in the population

```

// Populating Topic-Opinion pairs.
try {
// Omitted code not a part of this snippet, see the list of code C.15
//This line of code will only populate the topic-opinion pairs from app 1 (containing 181
reviews)!
    ResultSet rs = stmt.executeQuery("select p.reviewId as reviewId, p.subject as
subject, p.negation as negation, p.opinion as opinion, s.subjectCategory as
subjectCategory from pairsTable p, selectedpairsTable s where p.id = s.pairsId and
reviewId < 182;");
// Omitted code not a part of this snippet, see the list of code C.15

```

```
} catch (Exception e) {  
    System.out.println(e);  
}
```

However, it is observed that the bigger the ontology becomes, the slower the population is. It does release the memory if the population of one unit finishes. But once it resumes for another unit, the speed is far from the same as that at the beginning of the first unit.

Although these two methods do not reduce the memory resources significantly, they do speed up the process slightly and provide methods to manage the population in smaller units.

Clearly, both above methods are also helpful during the test phase for the ontology population code in a new project implementation.

3.6.4 Ontology population code structure

The OWL API is needed for ontology operations through Java. There are plenty of source code examples online that can be taken as the starting point of a Java project of managing ontology operations. A keyword search by “`OWLManager.createOWLOntologyManager();`” will bring them up in Google.

An `owlapi-bin.jar` file is needed in this population code, and it is recommended to include it in the Libraries folder of the Java project for convenience.

In this ontology populator, four ontology managers manage the population processes for apps, reviews, user requests, and topic-opinion pairs respectively. Each ontology manager reads the same ontology file that has been copied into the “Ontologies” folder in the Java project, and defines how the relationships among the input parameters should be set up, thereby the axioms are added into the ontology accordingly.

Among the above input parameters, a class can be defined beforehand if an object of such a class is necessary, such as the class “App”.

The main code is named as “ReviewDataExtraction” in this ontology populator. Inside the main method, there are four try-catch blocks dealing with the populations for apps, reviews, user requests, and topic-opinion pairs respectively in the same steps below.

- (1) Each try-catch block creates a specific ontology manager object first from the corresponding ontology manager defined above.
- (2) A connection is made to the MySQL database and queries the data from it.
- (3) Looping through the results and assigning the values of each record to the parameters that the ontology manager requires. Then inside the loop, call the ontology manager's method "populateReviewOntology" and supply it with the parameters.

The algorithm for complex ontology population that is mentioned previously in subsection 3.6.2 is implemented in the sequence of the database queries that are used in the step 2 above.

For the database connection code in step 2, there are plenty of example code online behind a keyword search by "Example to Connect Java Application with MySQL database".

A MySQL connector for Java is needed in order to make the connection work. It is flexible to place the connector anywhere as long as it is added into the Java Build Path correctly through the menu: Project – Properties.

Importantly, before querying the database, necessary privileges have to be granted to the role that is used in the database connection. An example is below:

Table 26 Granting privileges to the role in the database connection

MySQL 5.7	grant all on diabetevaluation.* to review@localhost identified by "review";
MySQL 8.0	Create user 'review'@'localhost' identified by 'review'; Grant all on drivingtheoryv2.* to 'review'@'localhost';

3.6.5 How to read existing individuals from the ontology if they exist

Inside the specific ontology manager, such as "TopicOpinionOntologyManager.java", if an individual will be populated only once, it is safe to pass an object of "Review" as one of the parameters and call the review object by the "getIndex()" method of the "Review" class when creating an IRI for this individual review: "IRI.create(iri + "#" + review.getIndex())".

However, above method will always create a new individual review for the ontology. If an individual review that has the same IRI exists in the ontology, it will be washed out by the new individual in this case.

The way to avoid this problem is to create the IRI of this review using the exact same IRI if it exists. In other words, the IRI of the same individual always remains the same regardless whether it is first created or it has existed before. To achieve this, “TopicOpinionOntologyManager.java” creates IRI for reviews by using their ids in the formats as they have been created, rather than calling the `getIndex()` method of the class: “`IRI.create(iri + "#review" + reviewId)`”.

3.6.6 Merging negation words with opinion words

If an opinion has a negation word that companies it, the negation word is concatenated as a prefix for the opinion in this ontology population. They are separated by a dot rather than a space because opinions are used as IRI directly in this ontology and ontologies do not tolerate spaces in names. There are two places in the code that achieve this feature in two steps.

The first place is in the main code when the opinion is populated. If the negation of this opinion is an empty value “null”, it is set as an empty string explicitly. This is necessary because it is currently unclear what the “null” value becomes later and how long it is if it is not caught here at the beginning.

The second place is in the “TopicOpinionOntologyManager.java”. If the length of a negation is 0, the opinion remains the same. Otherwise, the negation is treated as a prefix of the opinion separating by a dot.

3.6.7 Overcoming the data problems that crash the ontology

Ontologies are critical to data formats. Malformed data crash ontologies most of the time. Therefore it is better to handle malformed data earlier before they enter the database or flow to the database tables that will be populated.

The IRIs for apps, reviews and user requests use their ids in this ontology populator. The only types of individuals using their own names as IRIs are topics and opinions, therefore their names have to be free of malformed data.

The reason for topics and opinions using own names as IRIs is that ontologies subsume individuals by names. Although they are still different individuals, being subsumed and appearing together in the ontology facilitate the data management and queries.

There is an array “stopOpinions” introduced in 3.5.6 that can rule out irregular opinions in version one of the topic-opinion extractor. For “subject” (topics) and “negation”, the malformed data mainly contains “--”, two hyphens, which will definitely crash the ontology. Therefore, these strings are controlled from both two pieces of code in the Topic-Opinion extraction component. Moreover, lengthy strings are also ruled out at the same places. Lengthy strings of the sentences could crash the dependency parser. Too long strings of subjects, opinions, and negations that exceed the database definitions could also stop the code from running.

4 Results and evaluation

The purpose of this chapter is to report the results of the experiments and evaluate the performance. The objective of the evaluation is to answer the question of whether this prototype works. If the answer is yes, the hypothesis of this thesis is supported.

Section 4.1 introduces the evaluation methods. Section 4.2 reports the methods in the experimentation reflected through a checklist. Section 4.3 reports the performance results. An extra step on precision evaluation of the request elicitation component on the control group has to be taken. Therefore, the impact of the chosen sample size on the evaluation is further discussed in this section after that. Section 4.4 interprets the results and predicts what the best performance could be if all the fixable problems have been fixed. Section 4.5 quickly evaluates the negation words handling in this prototype. Section 4.6 reports the results of the ontology evaluated by pitfalls, presents some typical questions that users can query the GMAURO ontology, and provides the example results of Snap SPARQL queries.

4.1 Evaluation methods

Users may express requests, complaints, opinions, or simply use narrative sentences to tell stories that may have requests, complaints and opinions or may not. Both requests and opinions overlap with user complaints. But user requests seldom contain opinions, although they two do occasionally merge in one longer sentence. This prototype does not break such sentences into smaller units. Therefore, these sentences will only go through one component. Firstly, they are tested by the user request elicitor. If there is no request being picked up, they are tested by the topic-opinion extractor. Figure 29 depicts the general user review compositions and their relationships. User requests, complaints and opinions can be explicit or in implications.

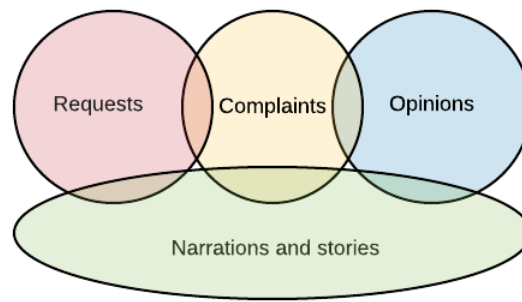


Figure 29 User review compositions

Because complaints overlap with requests and opinions, and complaints are not investigated in this version of the prototype, it is a bit difficult to evaluate the prototype performance with clear cuts. The policies used in this evaluation are as follows.

- 1) Allocating a complaint as a true positive result if it is elicited by a request linguistic rule;
- 2) Allocating a complaint as a true negative result if it is not elicited by a request linguistic rule.
- 3) Extracting opinions from a sentence containing complaints if it is not picked up by a request linguistic rule.

This evaluation mainly uses requirement engineers' expertise and especially with a focus on the explicit user requests and the opinions towards the specific aspects of the mobile apps. Debugging code from the components that use linguistic rules involved into the process in order to help the judgement and speedup the process. Taking the prototype with the version two of the topic-opinion extractor component, the system's performance will be compared among three datasets in the form of manual validation of the precision, recall, and F1 score of one sample set from each dataset.

All three datasets will be further investigated for the reasons of all FPs (false positives) and FNs (false negatives). The reasons will be classified into the common causes. Treatments will be given for each fixable cause. Then each fixable cause will be rectified in a future version. If the performance is not satisfactory, the prototype will be rerun against the three datasets after all the fixes. The performance will be manually recalculated for precision,

recall, and F1 score again. The new performance will be compared with the previous performance.

Based on the performance and whether the prototype needs the fixes, a conclusion will be drawn on whether the prototype works or has the adaptability for working performance.

The sample size of the user review number from each dataset is calculated as 383, and rounded up to 400. The calculation uses Cochran's sample size formula [113], which can be simplified as the formula below [114].

Sample Size Calculation:

$$\begin{aligned}\text{Sample Size} &= (\text{Distribution of } 50\%) / ((\text{Margin of Error}\% / \text{Confidence Level Score}) \\ &\text{Squared}) \\ &= (0.5 \times (1-0.5)) / ((0.05/1.96) \text{ Squared}) \\ &= 384.16\end{aligned}$$

Finite Population Correction:

$$\begin{aligned}\text{True Sample} &= (\text{Sample Size} \times \text{Population}) / (\text{Sample Size} + \text{Population} - 1) \\ &= 384.16 \times 90946 / (384.16 + 90946 - 1) \\ &= 382.55\end{aligned}$$

383 is a suitable number for normal distribution samples for the biggest dataset, dataset two that has 90946 user reviews. Rounding up to 400 user reviews for each dataset is safe and slightly over-sampled if the datasets are normal distributions. However, user reviews are in natural languages that are not considered as normal distribution due to their words and patterns. Moreover, user request data are sparse. Therefore, the random samples could produce different performance for user requests each time. In order to avoid this problem, all three samples are produced by a piece of code in its first successful random sampling for each dataset. This experiment uses the normal distribution statistic sample calculation formula. Though it is necessary to warn the readers that the user requests elicitation component's performance could fluctuate for different datasets under this sample size. Future researchers who take this framework are recommended to decide their sample sizes realistically based on their specific research questions and domain data.

The impact of this sample size on the evaluation is further discussed in the next section 4.3 after the report of the performance results and that an extra step on precision evaluation of the request elicitation component on the control group has to be taken.

A Python “random.sample” method is used to randomly sample a list of unique elements chosen from the population [115]. There is a piece of code that is dedicated to random sampling in this evaluation. This code first connects to the database where the corresponding population is located, and then performs a specified number of user reviews’ random sampling, and then integrates the data of the sample into the corresponding relationships among reviews, requests, topics and opinions, and finally presents the data in a spreadsheet per whole user reviews.

A key step before generating a spreadsheet in this code is the data integration of the corresponding relationships among the tables. This is accomplished through a full outer join. Since MySQL does not currently support direct full outer joins, this is achieved indirectly through a union between a left join and a right join. The details in this code are presented in the C.16 code in the list of code.

4.2 A checklist for the experimentation

A checklist [116] below is used to guide the experimentation and the later report of the results and evaluation.

Regarding the experiments:

- (1) Are there appropriate baselines that could be identified?
- (2) What data has to be gathered, and where from?
- (3) How will readers gather comparable data for themselves?
- (4) Is the data real? Is it sufficient in volume?
- (5) Will a domain expert be needed to interpret the results?
- (6) What are the likely limitations on the results?
- (7) Will the reported results be comprehensive or a selection? Will the selection be representative?

- (8) What enduring properties might be observed by other people attempting to validate the work with different hardware, data, and implementation?
- (9) Are the experiments feasible? Do you have the resources (time, machines, data, code, humans) required to undertake them to a reasonable standard?

Regarding the prototype:

- (10) Can the code be decomposed into components? How will the individual components be tested for correctness, and evaluated for significance?
- (11) How will you know that the code is correct?
- (12) Is the code going to be made publicly available?

Brief answers to the above checklist are as follows, then the detailed answers are reflected in the rest of this thesis.

- (1) **Are there appropriate baselines that could be identified?** This study is trying to provide a framework for the feasibility of solving the whole user review analysis problem that has no previously reported same solution. It is considered that there is no exact baseline to compare.

Gu and Kim's work [93] on aspect-opinion extraction is similar to the topic-opinion extractor in this thesis. The reported performance is also similar. But their evaluation focuses on the classified sentences in the "aspect evaluation" category and filters out other types, such as "praises" and "others". The topic-opinion extractor component in this thesis is evaluated in all sample reviews. Therefore, the basis for evaluating the performance of the two systems are different, and this potentially affects their direct comparability.

In addition, the performance of the topic-opinion extractor in this prototype is also different in the three datasets. Depending on different datasets, specific performance of this component can vary by as much as 7%. This can be spotted in the performance that was reported in section 4.3. These can be taken as reasons to understand that only a significant increase relative to the baseline, such as more than 8-10%, is worthy of an outperformance argument.

- (2) **What data has to be gathered, and where from?** The raw user reviews data are taken from Google Play using the first component, a set of web sourcing code.

- (3) **How will readers gather comparable data for themselves?** Readers are expected to be able to build their own web sourcing code after reading the subsection 3.2 of the Methods chapter. Moreover, the set of web sourcing code is provided in this submission. Therefore, the readers can use the code and adapt it to the up to date new web pages.
- (4) **Is the data real? Is it sufficient in volume?** The data are real. For the purpose of testing whether this prototype works, three datasets with 174,051 user reviews in total are considered as sufficient. This is a question that is rarely investigated by researchers from user review analysis area.

Martin *et al.* [117] surveyed a number of papers concerning “the app sampling problem for app store mining” and reported the related work datasets in the table II in their paper. According to their definition, the datasets used in this thesis are “Pa set” (apps with a proper subset of all submitted reviews). Given the fact that the mobile apps that were included into the datasets in this thesis all had passed certain inclusion criteria, it is safe to say that they are Pa sets. Taking the research cited by Martin *et al.* that used Pa sets, removing the duplicated datasets being used by different papers, and also removing the papers that use the most reviews and the least reviews (outliers), the number of the average user reviews being used by those papers is 85,645. This thesis uses 174,051 user reviews, which is approximately the same level. Therefore, it is appropriate to say that 174,051 user reviews are sufficient for the evaluation compared with other research.

However, it is important to emphasise that the work in this thesis is a starting point for other researchers to develop further, and the evaluation here is testing whether this prototype works. Besides the app store platforms are also raising the barriers for downloading the complete datasets, therefore, whether this thesis is using an “F set” (apps with all submitted reviews) is not a focus at this beginning point.

- (5) **Will a domain expert be needed to interpret the results?** Domain expertise involved in the interpretation of the results before this prototype was finalized. Mainly the results were interpreted by the author with the help of the debugging code. Therefore, the correctness of the output was judged by the author. Occasionally, the unsure results were consulted to the lecturers in the department. In future, it could be

done more objectively. Moreover, consultations were not limited in the judgements about the output. The discussions were also on the validity of the classifications of reviews, improvements of the results, and limitations of the best performance that the current prototype could achieve. For example, it is a disadvantage that the current prototype can only extract single-word topics and opinions. This has been suggested as one of the future research directions. Besides, requirement engineers' expertise is needed when interpreting the results in the evaluation. When interpreting the results, mainly two types of questions need to be answered. One question is whether the elicited phrases or short sentences are user requests. The other question is whether a user review contains opinions about an aspect of the mobile app. Based on the current results, which are the best outcome that could be accomplished within the available time frame, debugging code from the components that use linguistic rules involved into the process in order to help the judgement and speedup the process.

- (6) **What are the likely limitations on the results?** A random sample of 400 user reviews is used for each dataset. Because the user request data are sparse and not normal distribution, it is likely that the performance of the user request elicitation component is not sufficiently evaluated with the sample size of 400 user reviews.
- (7) **Will the reported results be comprehensive or a selection? Will the selection be representative?** The three datasets are selections of three different types of mobile app user reviews on Google Play, which are diabetes management, fitness training management, and driving theory mobile apps. These three datasets are not similar in language patterns, especially the third dataset are far away from dataset one and two in topic and language styles. It is arguable whether they are representative for all user reviews for other research questions such as mobile app metadata related questions. However, for proofing whether this prototype works for user requests and opinions, if all three sample sets produce remarkable precisions in the performance evaluation, it can be considered that they are representative.
- (8) **What enduring properties might be observed by other people attempting to validate the work with different hardware, data, and implementation?** This framework does not require special hardware and data. The way of linguistic rules being defined can be adaptable and evolve. The readers are recommended to valid the

linguistic rules if they change the training data that are set in the code. Same for their development of new rules, it is best to stick to one set of the same training data.

- (9) **Are the experiments feasible? Do you have the resources (time, machines, data, code, humans) required to undertake them to a reasonable standard?** The experiment is feasible for a competitive normal computer and does not require a special high performance computer. All the experiments are accomplished on a computer that has a 500 GB SSD hard disk and 32 GB RAM. Most of the code runs fast. The only two pieces of code that run slowly are the topic-opinion extractor that uses Stanford Parser and the ontology population code that populates all the reviews and resulting data from one dataset into a big ontology.
- (10) **Can the code be decomposed into components? How will the individual components be tested for correctness, and evaluated for significance?** The prototype is decomposed into five components that have been reported as in the previous 3.1 section. The coding correctness for each component must pass the unit testing before the experiments of the whole prototype take place. The experiments described in the next Results and evaluation chapter mainly test the performance of the two sets of linguistic rules.
- (11) **How will you know that the code is correct?** For the two components that use the linguistic rules, two pieces of separate debugging code have been developed to test correctness per single sentence. Therefore, each linguistic rule is tested as correct before integrating into the formal code dealing with the database.
- (12) **Is the code going to be made publicly available?** Yes, the code of the prototype, the debugging code and the ontology are made publicly available through Brookes RADAR.

4.3 Performance results

The three sample sets with 400 user reviews each are randomly sampled from the three datasets and produce performance as the numbers in Table 27. It is necessary to mention that they all are the first successful random samples when the code is set up properly.

Table 27 Performance of the first samples of three datasets

Dataset one (excluding the training set):

	Request:	Topic-Opinion:	General:
Precision:	92.9 %	89.2 %	89.5 %
Recall:	58.4 %	81.9 %	79.6 %
F1-score:	71.7 %	85.4 %	84.2 %

Dataset two:

	Request:	Topic-Opinion:	General:
Precision:	92.3 %	83.4 %	83.8 %
Recall:	53.3 %	77.4 %	75.7 %
F1-score:	67.6 %	80.3 %	79.5 %

Dataset three:

	Request:	Topic-Opinion:	General:
Precision:	92.3 %	92.1 %	92.1 %
Recall:	54.5 %	74.9 %	74.0 %
F1-score:	68.6 %	82.6 %	82.0 %

However, the control group (dataset three) produces higher precision for user request elicitation component than expected, therefore, another 2000 user reviews in this dataset are randomly sampled again, which are additionally and specifically for user request elicitation performance calculation for this dataset. The precision is reported as 87.2 %. This precision fluctuation is because of the user request data's sparsity. All the samples mentioned above are submitted with this thesis together in the evaluation folder.

As seen from the above performance that are reported in Table 27, the F1-scores for all three datasets are around or above 80%. The performance of this prototype proves that this

prototype works. Therefore, it is appropriate to conclude that the hypothesis of this thesis is supported by the performance of this prototype.

What is the impact of the chosen sample size on this evaluation? What measures could be taken to tackle the data sparsity problem, and what measures should not be?

The sample size of 400 user reviews has an impact on the evaluation due to the data sparsity and not being normal distribution. This is first reported in section 4.2 A checklist for the experimentation, point 6. The data sparsity problem is severe for the request elicitor component. As it is also reported in the previous section 4.1 that user reviews in natural languages are not normal distribution in their words and patterns. For a normal distribution without data sparsity problem, the 400 sample size is sufficient and slightly over-sampled. However, for the user review data in this evaluation, 400 sample size is not enough.

Could any measures such as averages and standard deviations of a few samples help? The conservative answer for such a question is No. The reason for this answer is that any trends and risks in the data might be repeated, accumulated, or even exaggerated in the results. Before such factors are investigated, it is difficult to predict the effectiveness of such measures. Moreover, the efforts that such measures cost are also no less than one bigger sample size. It could be reasonable to believe that one bigger sample provides better and more reliable results than such measures. However, the main difficulty for a bigger sample to be done is the available time and resources. This framework and the prototype are bigger than a normal workload for a typical PhD. Fortunately, the goal for this evaluation is to prove that this prototype works, which has been achieved in this thesis, rather than solving a precise practical research question that would need stricter and larger scaled sample evaluation.

A measure that could be taken to tackle the data sparsity problem in the sample size calculations is to reduce the value of Confidence Interval.

There is not sufficient literature that discusses the sample size calculations required to a measure of sparsity that could be followed straightforwardly. However, it may be the case that sparse data need bigger sample sizes. This can be achieved by adjusting the confidence interval value. Confidence interval is normally 5%, which can be reduced to a lower value in the case of dealing sparse data.

The best solution is bigger sample sizes. Within the realistic choices, the bigger, the better. Future researchers are also recommended to be realistic for their sample sizes in their evaluation. Their own goals on the performance evaluations should combine with their research questions and their practical usages.

The performance and the reasons for FPs and FNs of the topic-opinion extractor component will be discussed further in the next subsection.

4.4 Interpretation of the results

The hypothesis of this thesis is that such a framework that sets up a channel from raw user reviews to structured analysis data will be practical and usable. At the beginning of this Results and evaluation chapter, the objective of the evaluation has been set to answer the question whether this prototype works and therefore whether the hypothesis is supported.

From the performance of the random samples of the three datasets listed in Table 27, a conclusion can be drawn that this prototype system works, and furthermore it does not show the tendency of topic domain dependence and overfitting.

The topics of these three mobile app datasets are diabetes management, fitness training management and driving theory. All linguistic rules are summarized by the 6081 user reviews of one mobile app from the first dataset. The remaining user reviews of the first dataset together with the second and third datasets serve as the test datasets for this prototype system. Both the first dataset and the second dataset contribute to the collection of keywords and popular opinions lists. The third dataset acts as a control group and does not contribute to any linguistic rules, keyword collections, and popular opinion collections.

From the previous performance, the overall precision and recall of the first dataset have reached expectations. The performance of the second dataset showed a slight decline, but still remains at a level that is good enough to be accepted as working. The third dataset is relatively far from the previous two datasets in terms of topic and language patterns. Surprisingly, however, its overall performance has not declined. The reasons are discussed from the two components separately below.

For the performance of the user request elicitor, precision is expected, but recall is not ideal. This is because the training set is too small to cover more language patterns. The method of correcting the performance of this component is the same as that has been introduced in the previous section 3.4. Future researchers can make changes to the original rules or add new ones.

It is worth mentioning that the user requests data are very sparse, therefore a random sample in the size of 400 user reviews could produce various results each time. Although the precisions look stable across three datasets in Table 27, it is not appropriate to claim they are stable just by the samples with the size of 400 reviews. For example, the user request elicitor of the third data set, because it is a control group that never contributes to the linguistic rules, is likely to have inaccurate performance prediction. Therefore, another random sample containing 2000 user reviews was analysed. In this sample, as mentioned earlier, the precision drops to 87.2%.

In this new sample, 40% of FPs (4 out of 10) are generated by linguistic rule 99. Rule 99 is used to identify whether the user has expressed a request when using the modal "should". The performance of rule 99 is as expected in the training set, but there is a declining performance in the third data set. When investigating more user requests generated by rule 99, it is found that users are more likely talking about other topics that are away from expressing user requests, such as to persuade other users to use the current mobile app when using "should". At these times, the "get" and "download" vocabularies are used more in the language patterns. Therefore, in the next version of the improved rule 99, these words should be used to prevent such a sentence from passing through rule 99.

Table 28 Reasons for FPs and FNs in the three datasets

Reasons	Occurrences in Dataset1	Occurrences in Dataset2	Occurrences in Dataset3
a disabled rule	4	3	2
a missing keyword	6	14	5
a missing popular opinion	2	7	7
a missing relationship	12	17	5

a multiple meaning opinion	12	15	6
an abandoned keyword	1	1	0
an orphan opinion	5	1	1
complex sentence or lack of punctuation	7	6	3
entered a rule but failed the keyword test	83	65	52
miss-spelling related	15	23	27
multiple words are not supported	5	3	1
no obvious opinion term	7	4	6
not a good opinion in this context of use	35	23	1
not an opinion	1	0	0
tree parsing	11	14	20
WordNet over linking	1	0	2
total FNs + FPs	207	196	138
total records	1031	899	618

For the topic-opinion extractor component, due to its complexity, there are more reasons for FPs and FNs. Each FP and FN is analysed, and a cause and a corresponding treatment are determined in the evaluation process. The treatment may be an action, or “do nothing in this version”. If it is an action, this action has been tested on a new version of the debugging code.

Table 28 lists the various reasons for FNs and FPs and compares the occurrences in the three datasets. From this table, it can be seen that the data distribution of most of the reasons among the datasets is similar. But there are differences for individual reasons, such as "not a good opinion in this context of use". It also can be seen that the most records involve the reason “entered a rule but failed the keyword test”.

Table 29 provides a detailed explanation for each reason.

It is necessary to explain that these reasons are not completely exclusive and separable in logic. If it is possible to express specific detailed reasons, or because of the different

corresponding treatments, they will be distinguished as different reasons. The way in that they are separated is to help the analysis of the different reasons and the possible treatments.

Table 29 The explanations for the reasons of FNs and FPs

Reasons	Explanations for the reasons and treatments
a disabled rule	In the first version of the topic-opinion extractor, some rules that produced a small number of TPs were blocked for performance balance. However, in the second version of the topic-opinion extractor, this is less necessary. Therefore, it is appropriate to consider enabling some of those rules.
a missing keyword	The keywords list is the key of the concept mapping from the topics to the ontology. This list requires users of this system to maintain and increase or decrease according to the characteristics of the new dataset. If "a missing keyword" becomes the reason for a topic-opinion pair loss, adding this keyword to the keywords list will solve the problem.
a missing popular opinion	The list of popular opinions is extracted from the frequently used opinions of the first and second datasets. It is entirely possible that it is not complete. If "a missing popular opinion" becomes the reason for a topic-opinion pair loss, adding this opinion to the popular opinion list will solve the problem.
a missing relationship	A topic-opinion pair is dependent on language rules to establish relationships. These relationships were established through 6081 user reviews. It is unrealistic to rely on these 6081 user reviews to create a full set of language rules that apply to all datasets. If such a rule causes the loss of a pair of a topic and an opinion, the language rule needs to be added to the existing rules.
a multiple meaning opinion	An opinion is sometimes a polysemous word. When the word appears as an orphan opinion but has no meaning, then it is not welcomed by this topic-opinion extractor. At this time, it is

	necessary to introduce a contrary language rule to prevent it from appearing as an orphan opinion. The concept of this contrary language rule will be described in detail later in 5.2.2 in Discussion.
an abandoned keyword	A typical "an abandoned keyword" is "apps". Since users are more likely to point to other apps instead of the current app in meanings, the word was removed from the keywords list. But occasionally users will use it when praising the current app.
an orphan opinion	When "an orphan opinion" appears as the cause, it should be a real orphan opinion, but something goes wrong. It may be the negation or not a good match between the topic and the opinion. Usually this reason does not lead to an action of a treatment.
complex sentence or lack of punctuation	Stanford Dependency Parser makes mistakes when sentences are too complex or if several sentences are concatenated together but missing punctuation. These complicated sentences are sometimes not very long. But their complexity may be reflected in the unmarked turns of the semantics. This reason does not lead to an action of a treatment.
entered a rule but failed the keyword test	This reason usually leads to no action of treatment. This is because this reason usually refers to phrases like "very reliable". "Very reliable" is caught by rule 11, but such phrases rarely contain specific topics. Later in this section, whether to disable rules such as 11 in order to seek a better balance of performance will be discussed.
miss-spelling related	Spelling mistakes not only cause topics or opinions not to pass tests, but also often cause dependency trees errors. The majority of "miss-spelling related" records lead to no action. But occasionally the miss-spelled opinions are added to the popular opinions list, if this opinion is more common.

multiple words are not supported	Multiple words are phrases like "must have" and are not supported in the current prototype. The current prototype can only extract single-word topics and opinions.
no obvious opinion term	When users narrate a thing in a story-telling manner, they often express an opinion. But it can't be processed in the current prototype.
not a good opinion in this context of use	Sometimes some opinion terms do not help to judge the true opinions. For example, "honestly", "average", "pressure", "excel" and "values" appear in the context of mobile app user reviews, they are often not opinions. It is necessary to remove these opinions from the popular opinions list.
not an opinion	This reason rarely occurs. It refers to a word that is definitely not an opinion. For example, "care" appears in the phrase "customer care."
tree parsing	When this reason arises, it refers to the performance problems of the Stanford Parser. Either because of the abnormality of the sentence, or because of the redundant punctuation, it is quite certain that no action can be taken.
WordNet over linking	WordNet is used in this prototype system for lexical variance recognition of the keywords list. When the similarity threshold is 0.90, although the performance reaches a very good balance, occasionally the undesired words are still possibly linked to the topics. Because such a probability is already small, it is not necessary to take action for the time being.

Figure 30 provides a direct comparative view of the various causes of FNs and FPs among the three datasets in a percentage manner.

As can be seen from Figure 30, the most FNs and FPs records in the three datasets concentrate on "entered a rule but failed the keyword test". Among other reasons, the records of the three datasets are mostly similar. The differences in the three datasets mainly focus on "not a good opinion in this context of use". The first and second datasets are 3.4 percent and 2.6 percent,

respectively. Surprisingly, the third dataset as a control group was only 0.2 percent. In contrast, the records in the third dataset that are related to "miss-spelled related" and "tree parsing" reasons are significantly larger than the previous two datasets. These phenomena may be an indication caused by different language styles in dataset three.

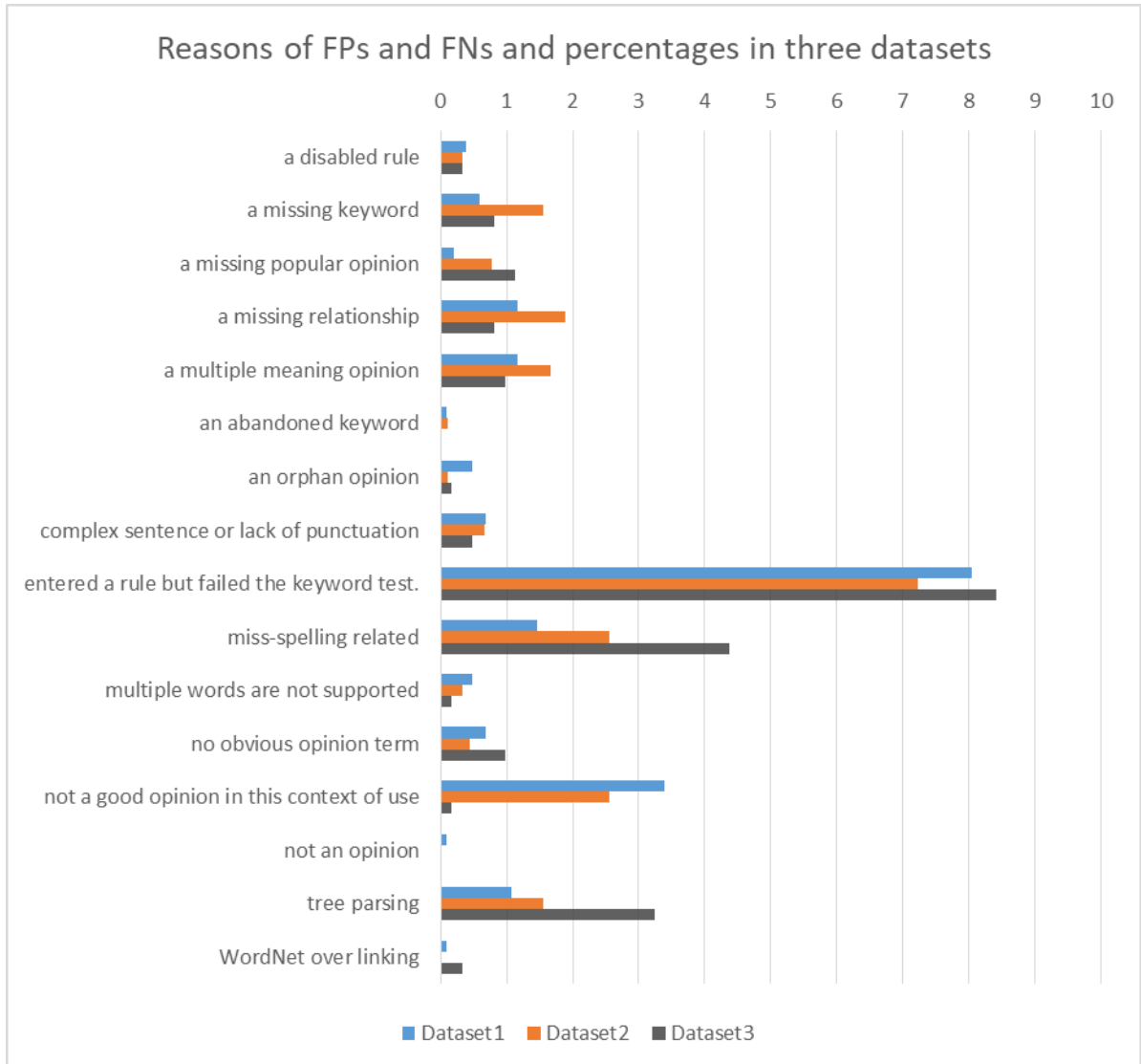


Figure 30 Reasons for FPs and FNs and the percentages in three datasets

Each cause is associated with one or more treatments. For the three datasets, the following three tables give the association and statistics between the reasons and the treatments. The statistics of these correspondences are helpful for analysing system performance and the

decision making for the next step. In order to facilitate the display of the tables, treatments are abbreviated as T1 to T7 below:

T1 - reactive a rule

T2 - add a new keyword

T3 - add a new opinion

T4 - add a new rule

T5 - add a new contrary rule

T6 - do nothing in this version

T7 - remove from popular opinions

Reasons of FPs and FNs and the treatments for dataset one:

Table 30 Association between the reasons and the treatments for Dataset one

Reasons	Occurrences	T1	T2	T3	T4	T5	T6	T7
a disabled rule	4	4						
a missing keyword	6		6					
a missing popular opinion	2			2				
a missing relationship	12				12			
a multiple meaning opinion	12					11	1	
an abandoned keyword	1						1	
an orphan opinion	5						5	
complex sentence or lack of punctuation	7						7	
entered a rule but failed the keyword test	83						83	
miss-spelling related	15			1			14	

multiple words are not supported	5						5	
no obvious opinion term	7						7	
not a good opinion in this context of use	35						5	30
not an opinion	1						1	
tree parsing	11						11	
WordNet over linking	1						1	
total FNs + FPs	207							
total records	1031							
total treatments		4	6	3	12	11	141	30
Percentages = total treatments / total records		0.39	0.58	0.29	1.16	1.07	13.68	2.91

Reasons of FPs and FNs and the treatments for dataset two:

Table 31 Association between the reasons and the treatments for Dataset two

Reasons	Occurrences	T1	T2	T3	T4	T5	T6	T7
a disabled rule	3	3						
a missing keyword	14		14					
a missing popular opinion	7			7				
a missing relationship	17				17			
a multiple meaning opinion	15					15		
an abandoned keyword	1						1	
an orphan opinion	1						1	
complex sentence or lack of punctuation	6						6	
entered a rule but failed the keyword test	65						65	

miss-spelling related	23			2			21	
multiple words are not supported	3						3	
no obvious opinion term	4						4	
not a good opinion in this context of use	23						1	22
not an opinion	0						0	
tree parsing	14						14	
WordNet over linking	0						0	
total FNs + FPs	192							
total records	899							
total treatments		3	14	9	17	15	116	22
Percentages = total treatments / total records		0.33	1.56	1.00	1.89	1.67	12.90	2.45

Reasons of FPs and FNs and the treatments for dataset three:

Table 32 Association between the reasons and the treatments for Dataset three

Reasons	Occurrences	T1	T2	T3	T4	T5	T6	T7
a disabled rule	2	2						
a missing keyword	5		5					
a missing popular opinion	7			7				
a missing relationship	5				5			
a multiple meaning opinion	6					6		
an abandoned keyword	0						0	
an orphan opinion	1						1	
complex sentence or lack of punctuation	3						3	

entered a rule but failed the keyword test	52						52	
miss-spelling related	27			5			22	
multiple words are not supported	1						1	
no obvious opinion term	6						6	
not a good opinion in this context of use	0							1
not an opinion	0						0	
tree parsing	20						20	
WordNet over linking	2						2	
total FNs + FPs	138							
total records	618							
total treatments		2	5	12	5	6	107	1
Percentages = total treatments / total records		0.32	0.81	1.94	0.81	0.97	17.31	0.16

From these above three tables, two questions arise.

- (1) If the listed treatment actions are implemented, how much can performance improve?
- (2) For treatment T6 "do nothing in this version", what could be improved in the future?

Answer to question (1):

T5 and T7 are mainly used to reduce FPs, so there is an increase in precision. T1, T2, T3, and T4 mainly act to reduce FNs, so there is an increase in recall. However, the impact of these treatments on performance is difficult to accurately predict with the simple addition of these numbers. This is because other factors such as negations and tree parsing are involved. Therefore, the following effects on performance can only be understood as the maximum impact under ideal conditions. At the same time, since this is based on the calculation of random samples, its accuracy will also be challenged randomly in another random sample.

Table 33 Maximum performance impact prediction after the fixes for three datasets

Maximum performance impact prediction	Dataset 1	Dataset 2	Dataset 3
precision increase prediction if all fixes work	+ 3.98	+ 4.12	+ 1.13
recall increase prediction if all fixes work	+ 2.42	+ 4.78	+ 3.88

From the possible performance improvement predictions that are made in Table 33, it is indicated that there is a limit for such a user review analysis system on the best performance. This limit can vary among different datasets that have different language characters. However, users always express information in different ways. Some users address concerns for new user requirements. Some users express opinions to existing requirements that have been implemented in the software products they are using. Some of the users tend to narrate stories that may carry an opinion to the software but are hard to extract explicitly, or even the stories do not contain opinions. Needless to mention that a portion of user reviews could be totally irrelevant to the questions that requirement engineers intend to answer. In the context of the variety of user reviews, it is an understandable fact that a user review analysis system has a limit in performance that relates to the research questions and datasets.

Answer to question (2):

With respect to treatment 6, "do nothing in this version", in each sample set, half or more than half of the records are related to "entered a rule but failed the keyword test". Among these related records, the rule with the most associated records is rule 11. The definition of rule 11 is:

[11, 'advmod', 'RB', 'JJ', 'child'].

If disable rule 11, what effect will it have on performance?

The answer is a deterministic increase for recall, but the impact can be complicated for precision. The following illustration is based on the training set.

First, the impact on recall is explored.

The training set contains 6081 user reviews for a mobile app. Rule 11 picks up only one record from these 6081 user reviews. However, if disable rule 11, there will be considerable orphan opinions identified. As described in the previous subsection 3.5.5 Figure 22, there is a significant decline in the recognition rate of orphan opinions in the two-word sentences. Thus, the sentences of two words in the training set are specifically counted for potential orphan opinions that could be identified if disabling rule 11.

In the training set, there are 963 two-word sentences, of which 635 are recognized by the system, and 328 sentences are not recognized, so the recognition rate is 65.94%. Among the 328 sentences, about 86 are unidentified orphaned opinions due to rule 11. If the 86 orphan opinions are all identified, the recognition rate may rise to 74.87%, which is a considerable 8.93% increase. Sentences of other lengths may also grow, but the magnitude will not be higher than the two-word sentences, because other linguistic rules and factors may involve. Therefore, the overall growth of recall should be lower than the two-word sentences.

Then the impact on precision is discussed below.

Rule 11 is a relationship that contains a pair of an adverb and an adjective and takes the adjective as the opinion. The effect on recall in the previous discussion is mainly about the adjective. For the impact on precision, attention must be given to the adverb. This is because if rule 11 is disabled, both words may become orphan opinions.

But the problem at this time is that the adverbs bound by rule 11 are not opinions. If they are released from rule 11, and are not recognized by other rules, they will be judged whether they are orphan opinions. At this point, if the adverb is also in the popular opinions list, it will become an orphan opinion according to the current code logic. This is not the expected result. Because when they exist independently as orphan opinions, they are either not opinions or their meanings change.

Using a hard coding method to rule out this scenario is not a good approach. There are many of such adverbs, for example, in the training set: "enough", "fairly", "honestly", "immensely", "importantly", "pretty", "super", "tremendously", "truly", "well", "wonderfully".

Moreover, it is important to mention that rule 11 and some other "advmod" rules are actually more about the extent to which an opinion is described. Therefore, retaining these rules and making specific development plans in the future will be an interesting research question.

In summary, the experimental results of the three datasets confirm that the performance of the prototype system is acceptable, and does not show the dependency related to the topic domains of mobile applications and the problem of overfitting.

In the Discussion chapter, the adaptability, flexibility, advantages and disadvantages of each component, and future research directions will be further explored.

4.5 A quick evaluation for the negation words handling

When users express their opinions, negations are sometimes mixed in. As described by Wiegand *et al.*'s survey [15], the processing of negation is complicated. In this prototype, the processing of negations is limited to the extraction of explicit negation words. Namely, no implicit negation, shifters or irony are considered. Since topics and opinions are extracted by traversing the dependency trees, the problem is greatly simplified. In the topic-opinion extractor component version two, negation words are roughly divided into three scenarios, the parent with "dep" relationship, a sibling with "neg" relationship, or a child with "neg" relationship. This was introduced in a previous section 3.5.3.

However, because dependency trees do not always produce regular and stable results, negation words sometimes appear in other relationships. Sometimes, the sentence structure is too complicated, or several sentences are connected together without punctuation. These situations can make negations difficult to handle. Therefore, the current negation words processing in this prototype is not perfect. Nevertheless, in order to understand the situation, the performance of negation handling is still evaluated and reported here.

Table 34 Performance of negation handling in three datasets

	Dataset one (excluded training set)	Dataset two	Dataset three (400 reviews)	Dataset three (2000 reviews)
Precision	82.05 %	87.10 %	36.36 %	63.83 %

Recall	91.43 %	67.50 %	50.00 %	49.18 %
F1-score	86.49 %	76.06 %	42.11 %	55.56 %

The previous three sample sets from each of the three datasets are reused here. These are the first three performance columns in Table 34. However, negation words are very sparse in the third dataset. This resulted in the sample of 400 reviews with only 4 true positives, 7 false positives, and 4 false negatives. Such little data is hard to trust. Therefore, another sample set containing 2000 reviews was extracted from dataset three. The performance is reported in the last column of Table 34. However, it is believed that due to the sparsity of the negation data in dataset three, the impact on the overall performance is relatively small. The four negation evaluation spreadsheets are in the “evaluation” folder of the submission with this thesis.

Unlike the previous evaluation of topic-opinion pairs, the evaluation of this subsection focuses on the extraction of topic-opinion pairs related to negation words. Topic-opinion pairs that are not related to explicit negation words are ignored in this evaluation. Due to these differences, the definitions of the various results are given below.

Table 35 Definitions of various results in negation evaluation

TP	This consists of an opinion consisting of an explicit negation word and an opinion word, and a topic word that is modified by the opinion. If the previous sample evaluation spreadsheets were reused, all TP topic-opinion pairs containing explicit negations would be TPs here.
FP	This consists of an explicit negation word, an opinion word, and a topic word being modified, but these three do not constitute a reasonable combination in the current sentence. This includes the case where the negation and opinion words are not a matched opinion, and it also includes the mismatch between the opinion and the topic. If the previous sample evaluation spreadsheets were reused, all FP topic-opinion pairs containing explicit negations would be FPs here.

FN	<p>This means a valid combination that has an explicit negation word, a matched opinion word, and a matched topic word, is not presented. This includes the case where a pair of matching opinion words and topic words are presented but the negation word in the opinion is missing, also includes the case where all three components are missing, and the case where narrative sentences to tell stories with opinions that contain negation words but do not contain explicit opinion words.</p> <p>If the previous sample evaluation spreadsheets were reused, all FP and FN topic-opinion pairs that do not contain explicit negations but there are explicit negation words in the sentences should be verified individually for whether they are FNs here. FN narrative sentences to tell stories with opinions that contain negation words should also be verified.</p> <p>In the case of a new evaluation spreadsheet, three cases need to be considered: there are opinion word and topic word pairs extracted and there is an un-extracted negation word in the sentence that forms part of this opinion; no part is extracted, but there is such a reasonable combination in the sentence; narrative sentences to tell stories with opinions that contain negation words but do not contain explicit opinion words.</p>
TN	<p>All other cases constitute TNs. In the performance calculation formulas currently used, only precision, recall, and F1-score are calculated, therefore the number of TNs does not affect performance.</p>

The current negation words being captured are either having a “neg” relationship, or among negation words “no”, “not”, “n’t” or “never” that have a “dep” relationship. The negation words “nor” and “none” rarely appear in an opinion that describes a topic of the mobile apps. If they appear in such an opinion and have a relationship that is not “neg”, they constitute FNs in this evaluation.

It is particularly worth noting that "nothing" is in the popular opinions list therefore is handled directly as an opinion, which does not require negation handling.

The process to produce the evaluation data from the previous three sample sets spreadsheets (the former three performance columns in Table 34) is below:

Step 1: mark sentences that contain explicit negation words “no”, “not”, “n’t” or “never” as value “1” at a new column “isNegation”.

The current MS Excel supports the function to check whether an excel cell contains specific text. An example formula, which helps check whether a review sentence contains “no”, “not”, “n’t” or “never” and assign a value “1” if it does or a value “0” if it doesn’t, is below:

Table 36 An Excel formula to check whether a review sentence contains a negation word

<pre>= IF(OR(ISNUMBER(SEARCH("not", D2)), ISNUMBER(SEARCH("n't", D2)), ISNUMBER(SEARCH("never", D2)), ISNUMBER(SEARCH("no", D2))), 1, 0)</pre>
--

In this formula, column D is where the review sentences are. If this formula is applied to the column “isNegation”, a manual process is necessary to rule out some non-negation phrases such as “not only”, “no matter”, and “no bother”. It is necessary to point out that sentences that contain “nor”, “none” and “nothing” will also return because they match “no”.

Step 2: find sentences that have value “1” at column “isNegation”, and non-blank values at column “negation”, then copy the values from column “pairResult” to column “negationResult” (a new column).

Step 3: find sentences that have value “1” at column “isNegation”, blank values at column “negation”, and “FN” or “FP” values at column “pairResult”, then decide one by one whether the value at column “negationResult” should be “FN”.

The process to produce the evaluation data from the new 2000 reviews sample set spreadsheet (the last column in Table 34) is below:

Step 1: mark sentences that contain explicit negation words “no”, “not”, “n’t” or “never” as value “1” at a new column “isNegation”. If a formula that is similar to Table 36 is applied to the column “isNegation”, a manual process is necessary to rule out some non-negation phrases such as “not only”, “no matter”, and “no bother”.

Step 2: find sentences that have value “1” at column “isNegation”, and non-blank values at column “negation”, then decide one by one whether the value at column “negationResult” should be “TP” or “FP”.

Step 3: find sentences that have value “1” at column “isNegation”, blank values at column “negation”, then decide one by one whether the value at column “negationResult” should be “FN”.

Why could such a simple negation words handling approach achieve acceptable results? Why are the results in dataset three not as good as dataset one and two?

The reason for such acceptable results in dataset one and two is the robustness of dependency trees. For regular sentences, Stanford Parser performs better in negation words parsing. Different sub-clauses are parsed into different branches as subtrees. Subtrees effectively work as the best spans for negation words handling.

The answer for the decreased performance of negation words handling in dataset three could be the high percentage of irregular sentences due to the different language patterns in this dataset. For irregular sentences, Stanford Parser gets confused easier. This could result in misshaped dependency trees, which blur the spans for negations. Irregular sentences are typified by miss-spellings, grammatical violations, and missing punctuation between consecutive several sentences. In the previous subsection 4.4, it is observed that dataset three has more false results in miss-spelling related errors and tree parsing errors. This is depicted in Figure 30 when comparing the false results for different reasons across three datasets.

It is an interesting question for how to better handle negations in this framework that performs more stably across different topic domains. This could mean that setting up a mechanism that can recognize spans for negations is necessary. Punctuation should play an important role in the mechanism. Negation spans are difficult to recognize when punctuation is missing. Then a proper size of a window around the negation words could complement the problem.

4.6 Several SPARQL queries about some applications of the GMAURO ontology

Ontology evaluation is a very technical issue. A number of studies have made outstanding contributions in this area. However, the available tools that provide support to knowledge engineers are few [118]. The ontology can be evaluated from the perspectives of pitfalls, metrics, and competence questions (CQ). Ontology Pitfall Scanner (OOPS!) is a good tool to evaluate ontologies by pitfalls [119]. Some scholars have studied the ontology based on

metrics [120]–[123], criteria [124] or metrics with an emphasis towards usability [118]. CQ is another method of ontology evaluation based on natural language expressions. Wisniewski *et al.* [125] summarized a set of CQ patterns and showed some examples of converting CQ to SPARQL queries.

The populated GMAURO with dataset one data was once attempted to be uploaded to OOPS, but was not successfully evaluated. This huge size ontology crashed OOPS. The server returned this message: “Bad Gateway. The proxy server received an invalid response from an upstream server.” It is understood that this is due to the large size of the populated GMAURO.

An unpopulated GMAURO ontology definition was submitted to OOPS and evaluated by pitfalls. OOPS has three levels of results: Critical, Important, and Minor. GMAURO got three types of “Important” and two types of “Minor” pitfalls diagnosed. Figure 31 shows the brief results that OOPS reports for GMAURO.

Results for P08: Missing annotations.	89 cases Minor 🟡
Results for P11: Missing domain or range in properties.	32 cases Important 🟠
Results for P13: Inverse relationships not explicitly declared.	9 cases Minor 🟡
Results for P30: Equivalent classes not explicitly declared.	1 case Important 🟠
Results for P41: No license declared.	ontology* Important 🟠

Figure 31 The pitfalls that OOPS! found for GMAURO

The detailed results are in the Appendix F OOPS!’s evaluation result for GMAURO. It is necessary to report that the properties that were diagnosed to have missing domain or range in P11 are either properties that are not in use in this version or data properties whose data types have been defined in their definitions with the classes that they work for. The cases that were classified as P13 are not currently in use and not necessary to have inverse relationships. The case that was identified as equivalent classes in P30 is actually two different classes.

However, how to properly evaluate GMAURO ontology is not the primary issue of the current prototype system. The popular ontology assessment methods are very technical. This prototype is still in its initial stage, and there is still much room for improvement in the ontology used. Due to the size of the populated ontology, online assessment tools can easily

collapse. A typical CQ is often concerned with the technical structure of the ontology. Currently, the focus of this section is on SPARQL queries for several usage scenarios that users may be concerned about.

GMAURO is a relatively shallow and straightforward ontology. It may be the first ontology in the field of mobile app user reviews. It still contains some “Attribute” classes that aim for further development for more complex questions that are not answered in this version. In this section, 12 questions that users may like to ask are presented below and efforts have been made to answer these questions using SPARQL.

Q1. Regarding one aspect, what topics have people mentioned?

Q2. What are the opinions of users on topics in this category?

Q3. What are the opinions of people on a single topic?

Q4. How many reviews are there in each mobile app?

Q5. What topics did users mention in a mobile app?

Q6. Which topics do people have most opinions about in a certain category, such as `UserInterface`, `Feature`, `Function`, etc.?

Q7. For all topics, how many opinions are on each topic?

Q8. For all topics, how many people mentioned each topic?

Q9. How many people have expressed opinions on a single topic?

Q10. What are the opinions on a single topic category, and how many people think about each opinion?

Q11. In the reviews that mention topics in a certain category, how many people have left positive reviews for each topic (or how many people have left negative reviews, and how many people are neutral)?

Q12. In a mobile phone application, how many people have left positive reviews (or how many people have left negative reviews, and how many people are neutral)?

Using Snap SPARQL Query is recommended.

Because the populated ontology of each data set is large, Protégé’s SPARQL Query tab is difficult to open at an acceptable speed. This has been explained in the 3.6.1 Ontology design section. To answer the above questions, it is necessary to recommend Snap SPARQL Query.

Snap SPARQL Query is not a default tab in Protégé. To use it, it is necessary to create a new tab, then find the Snap SPARQL Query from Window → Views → Query views and drop it into the new tab that is just created.

Snap SPARQL Query is designed for queries involving inferred knowledge. However, currently, a query returns an empty result if a reasoner is selected but not started. In order to get results, reasoning must be either started, or disabled [126].

However, if the ontology is very big, starting a reasoner can take an unacceptable long time, which is the case for all three datasets that are used in this evaluation. Therefore, the example answers below are run when the reasoners are disabled (Reasoner → None). In the current version of GMAURO ontology, there is only inference for the polarities of reviews, which have been replaced with calculations through SPARQL and answered in the questions Q11 and Q12 above.

Example SPARQL queries are provided for the above questions. They are listed with the screenshots for the answers returned by the Snap SPARQL Queries together in Appendix G Answers to the SPARQL queries about the applications of the GMAURO ontology.

The current version of GMAURO does not fully answer all the above questions. Question 5 takes an unacceptable long time to answer, and the answer for question 10 is not completely correct.

The execution time is also different depending on the calculations and object properties involved in these issues. Taking Data Set 1 as an example, the following table lists the execution time (in milliseconds) for each query example.

Table 37 Time taken by Snap SPARQL when answering the questions

Questions	Times taken to answer in milliseconds
Q1. Regarding one aspect, what topics have people mentioned?	6 ms

Q2. What are the opinions of users on topics in this category?	876 ms
Q3. What are the opinions of people on a single topic?	545 ms
Q4. How many reviews are there in each mobile app?	8360 ms
Q5. What topics did users mention in a mobile app?	Unacceptably slow
Q6. Which topics do people have most opinions about in certain category, such as UserInterface, Feature, Function, etc.?	581 ms
Q7. For all topics, how many opinions are on each topic?	74805 ms
Q8. For all topics, how many people mentioned about each topic?	960353 ms
Q9. How many people have expressed opinions on a single topic?	406 ms
Q10. What are the opinions on a single topic category, and how many people think about each opinion?	997850 ms
Q11. In the reviews that mention topics in a certain category, how many people have left positive reviews for each topic (or how many people have left negative reviews, and how many people are neutral)?	620 ms
Q12. In a mobile phone application, how many people have left positive reviews (or how many people have left negative reviews, and how many people are neutral)?	317 ms

As it can be seen from this Table 37, even a Snap SPARQL query can take quite a while when querying a large ontology. Compared with database queries, it is clear that querying ontology has no advantage in speed.

It is also necessary to analyze the reasons for the questions that GMAURO cannot answer.

In Table 37, Q5 semantically defines a question that GMAURO should be able to answer. However, because three object properties are used in the definition, the answer speed of GMAURO is unacceptable on a normal computer, therefore it has to be forced to quit.

The number of object properties in the remaining questions is less than or equal to two. These queries to GMAURO can be performed. It can be seen from this that for a GMAURO query,

the number of object attributes used increases, which may lead to an increase in the complexity of the query, thereby significantly increasing the query time.

In addition, the answer to Q10 is only partially correct. Question 10 is "What are the opinions on a single topic category, and how many people think about each opinion?" The answer to the first half of the question "What are the opinions on a single topic category" is correct, which is in the format of specific opinions following specific topics in the category. However, the answer to the latter half of "how many people think about each opinion" is for the entire dataset, not what is expected for the number of reviews containing each opinion for each topic in this category.

Ideally in a programming language, this is easy to achieve through a "for" loop that could loop through the results from the first half question, and then find the number of reviews for each of the combinations within the loop. However, this is not achievable through a SPARQL query because SPARQL does not support iteration. This suggests that competence questions actually have limits for what types of questions that can be asked, although such limitations are seldom reported in CQ related research papers.

There are other questions that GMAURO cannot answer at present. For example, "what color do users like the most about the elements in the user interface"? To answer this question, two links are needed. One link is between the user's opinion and the color, which is currently what GMAURO can establish. The other link is between the color and the element in the UI, which GMAURO has attempted to establish through the relationships between the "Attribute" classes and the "Category" classes. Yet "Attribute" classes are not currently supported by this prototype.

It is arguable whether it is worth using ontology. The answer varies for different scenarios.

Ontology has many benefits [127]. The first benefit is that automated reasoning can be enabled in an ontology. However, in the case of this prototype, the ontologies are too big for the reasoners to start on a normal computer. If the reasoning is not complicated and can be easily replaced by calculations through SPARQL, as in this GMAURO, reasoning is not necessary.

The second benefit of ontology is that the concepts and relationships are presented in the ways that are close to the way that humans perceive. It is then easy to navigate between concepts. This benefit is substantial. However, for researchers in computing, it is less significant than for normal users. Especially in the context of queries, only defined concepts and relationships can be queried. That is, if the definition of the relationships between concepts have not taken into account the use case of the query, it may be difficult to query the ontology, as in the case of question 10 above. Needless to say, the situation in question 10 can be easily solved in a relational database. Moreover, the case of question 10 also highlights the incapability of SPARQL in complex queries.

Another benefit of ontology is that it can represent more data formats than relational databases. For example, individuals with the same IRI will be grouped into one concept. This is very convenient for the management of concepts and relationships. However, for an ontology that is populated with a large amount of individual data, it is more difficult to count the complete number of individuals under a concept than a relational database.

In general, the benefits of ontology are real. But in different situations, the benefits of ontology are different. The advantages of reasoning in concepts and relationships make ontology more suitable for scientific research. In the use cases where there are large amounts of individual data and the concepts are simple, the advantages of the ontology are not significant.

These will be interesting research questions, such as how to effectively evaluate the GMAURO ontology, how to improve the design of GMAURO, and how to improve the queries to GMAURO.

At the end of this section, the question 1 and question 2 in the previous common question list Table 37 were answered using all three datasets. The generated data (after some string pruning) is also presented here.

Question 1: Regarding UserInterface, what topics have people mentioned?

Table 38 An example of Snap SPARQL Query for Question 1.

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

```

PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX generalMobileAppUserReviewOntologyVersion14:
<http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserRe
viewOntologyVersion14#>
SELECT ?category WHERE {
  ?category rdfs:type generalMobileAppUserReviewOntologyVersion14:UserInterface.
}
ORDER BY ASC(?category)

```

Table 39 The answer to Question 1 from dataset one

activities; activity; background; backgrounds; button; buttons; color; colors; crashes; display; displays; fields; font; fonts; giggling; green; grey; gui; icon; icons; interface; interfaces; layout; light; material; message; messages; noise; noises; notifications; perspectives; presentation; ray; red; screen; screens; scroll; search; setting; settings; size; sizes; sound; sounds; stuff; symbol; symbols; ui; view; voice

Table 40 The answer to Question 1 from dataset two

activities; activity; background; backgrounds; black; bread; burns; button; buttons; clash; color; colors; crash; crashes; creaks; crunch; crunches; dark; demo; demos; display; displays; flag; font; fonts; grumble; gui; icon; icons; interface; interfaces; layout; light; lights; material; materials; message; messages; noise; noises; notification; notifications; orange; perspective; pomp; popup; popups; presentation; screen; screens; scroll; search; seeking; setting; settings; size; sizes; sound; sounds; splash; stuff; sunlight; symbols; ui; view; views; voice; voices; white

Table 41 The answer to Question 1 from dataset three

activities; background; boom; button; buttons; color; colors; colour; crash; crunch; display; flag; flags; green; gui; icon; icons; interface; layout; light; lights; material; materials; message; noise; noises; notification; notifications; perspective; presentation; red; screen; search; setting; settings; size; sound; sounds; splattering; stuff; ui; view; views; voice

Question 2: What are the opinions of users on topics in this UserInterface category?

Table 42 An example of Snap SPARQL Query for Question 2.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX generalMobileAppUserReviewOntologyVersion14:
<http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserRe
viewOntologyVersion14#>
SELECT ?category ?opinions WHERE {
?category rdf:type generalMobileAppUserReviewOntologyVersion14:UserInterface.
?opinions rdf:type generalMobileAppUserReviewOntologyVersion14:Opinion.
?opinions generalMobileAppUserReviewOntologyVersion14:opinionIsRegardingSubject
?category
}
ORDER BY ASC(?category)
```

Table 43 The answer to Question 2 from dataset one

different activities ; love activities ; active activities ; busy activity ; lighter background ; outdated background ; prefer background ; wish background ; better backgrounds ; save button ; simple button ; big button ; uninstall button ; hitting button ; big buttons ; intuitive buttons ; lighter color ; new color ; blind color ; same color ; like color ; suggest color ; hard color ; new colors ; bright colors ; help colors ; love colors ; low crashes ; want display ; new display ; helpful display ; simple display ; great display ; nice display ; fails display ; active display ; current display ; wrong displays ; new fields ; different fields ; many fields ; empty fields ; pertinent fields ; comprehensive fields ; tiny font ; big font ; small font ; light font ; hard font ; tiny fonts ; difficult fonts ; small fonts ; adjustable fonts ; annoying giggling ; little giggling ; invisible green ; light grey ; pleasant gui ; not.busy gui ; awesome gui ; easy gui ; clean gui ; nice gui ; intuitive gui ; ugly gui ; save icon ; fine icon ; love icon ; delete icon ; loving icons ; pleasing icons ; annoying icons ; easier interface ; fantastic interface ; n't.streamlined interface ; lovely interface ; slow interface ; new interface ; convenient interface ; cramped interface ; super interface ; helps interface

; not.better interface ; elegant interface ; better interface ; clearer interface ; solid interface ; crowded interface ; bad interface ; easy interface ; complex interface ; clear interface ; clean interface ; sucks interface ; simple interface ; great interface ; consistent interface ; useful interface ; horrible interface ; old interface ; encourages interface ; free interface ; not.sophisticated interface ; tedious interface ; editable interface ; problematic interface ; friendly interface ; usefull interface ; cool interface ; not.user-friendly interface ; love interface ; nice interface ; easiest interface ; works interface ; worse interface ; brilliant interface ; appealing interface ; nicer interface ; excellent interface ; complicated interface ; intuitive interface ; ugly interface ; responsive interface ; like interface ; motivational interface ; playful interface ; hate interface ; good interface ; rich interface ; needs interface ; current interface ; clunky interface ; need interface ; different interfaces ; better interfaces ; new layout ; organized layout ; normal layout ; better layout ; clear layout ; clean layout ; complete layout ; great layout ; thoughtful layout ; beautiful layout ; best layout ; love layout ; nice layout ; appealing layout ; poor layout ; excellent layout ; like layout ; simpler layout ; good layout ; high light ; new material ; crazy message ; new messages ; annoying messages ; great messages ; n't.need messages ; old messages ; love messages ; same messages ; horrible noise ; assistant noise ; love noise ; little noise ; funny noises ; annoying noises ; silly noises ; nice notifications ; many perspectives ; cramped presentation ; easy presentation ; clear presentation ; great presentation ; unique presentation ; good presentation ; easy ray ; big red ; good red ; broken screen ; similar screen ; n't.tiny screen ; easy screen ; simple screen ; annoying screen ; big screen ; great screen ; small screen ; bigger screen ; love screen ; nice screen ; efficient screen ; larger screen ; pops screen ; same screen ; losing screen ; opens screen ; large screen ; first screen ; hard screen ; slow screens ; different screens ; difficult screens ; smaller screens ; clear screens ; larger screens ; no.busy screens ; slow scroll ; easy scroll ; not.well scroll ; able scroll ; overwhelming scroll ; like scroll ; forever scroll ; allow scroll ; want search ; new search ; easy search ; help search ; love search ; like search ; simple setting ; handy setting ; optimised setting ; no.problem setting ; help setting ; lose setting ; flexible settings ; n't.understand settings ; many settings ; help settings ; love settings ; perfect settings ; new size ; increase size ; small size ; readable size ; like size ; different sizes ; obnoxious sound ; n't.hard sound ; happy sound ; love sound ; crazy sound ; neat sound ; little sound ; funny sounds ; great sounds ; sorry sounds ; freak sounds ; silly sounds ; joyful sounds ; cool sounds ; love sounds ; good sounds ; want stuff ; new stuff ; helps stuff ; amazing stuff ; important stuff ; quick stuff ; great stuff ; n't.need stuff ; old stuff ; extra stuff ; cool stuff ; regular stuff ; legal stuff ; good stuff ; need stuff ; small symbol ; like symbols ; busy ui ; new ui ; latest ui ; okay ui ; loving ui ; amazing ui ; better ui ; crowded ui ; awesome ui ; easy ui ; simple ui ; great ui ; useful ui ; friendly ui ; love ui ; nice ui ; not.best ui ; works ui ; appealing ui ; improved ui ; nicer ui ; fast ui ; excellent ui ; outstanding ui ; intuitive ui ; good ui ; clunky ui ; helps view ; helpful view ; better view ; easy view ; clear view ; handy view ; confusing

view ; big view ; able view ; quick view ; great view ; small view ; n't.allow view ; limited view ; good view ; helpful voice ;

Table 44 The answer to Question 2 from dataset two

n't.save activities ; flawless activities ; different activities ; save activities ; kudos activities ; n't.want activities ; disappointed activities ; awesome activities ; easy activities ; able activities ; old activities ; streamlines activities ; faulty activities ; brilliant activities ; useless activities ; intense activities ; relevant activities ; correct activities ; good activities ; healthy activities ; new activities ; similar activities ; favourite activities ; helpful activities ; normal activities ; not.important activities ; not.accurate activities ; recommended activities ; keep activities ; never.works activities ; great activities ; useful activities ; many activities ; love activities ; fail activities ; dominate activities ; positive activities ; not.able activities ; n't.public activities ; perfect activities ; not.great activities ; favorite activities ; n't.save activity ; pointless activity ; not.current activity ; latest activity ; save activity ; beastly activity ; n't.quit activity ; helps activity ; primary activity ; hectic activity ; not.save activity ; solid activity ; honest activity ; not.good activity ; confusing activity ; complete activity ; not.interesting activity ; old activity ; encourages activity ; tedious activity ; n't.functioning activity ; general activity ; solitary activity ; crashes activity ; usefull activity ; cool activity ; decent activity ; fails activity ; not.delete activity ; properly activity ; n't.different activity ; ready activity ; correct activity ; missing activity ; good activity ; frustrating activity ; needs activity ; healthy activity ; new activity ; difficult activity ; encourage activity ; rubbish activity ; crashing activity ; keep activity ; simple activity ; annoying activity ; stuck activity ; wrong activity ; great activity ; saving activity ; useful activity ; many activity ; best activity ; recommend activity ; n't.bother activity ; TRUE activity ; delete activity ; nice activity ; lose activity ; n't.delete activity ; not.able activity ; moving activity ; excellent activity ; opening activity ; first activity ; no.save activity ; busy background ; transparent background ; serious background ; annoying background ; unique background ; different backgrounds ; annoying black ; buy bread ; fat burns ; love burns ; different button ; save button ; kudos button ; primary button ; tiny button ; clearer button ; easy button ; restore button ; wonky button ; crashes button ; bigger button ; useless button ; same button ; not.accessible button ; like button ; little button ; good button ; needs button ; keeping button ; allows button ; difficult button ; not.working button ; big button ; wrong button ; great button ; freezes button ; hitting button ; prominent button ; small button ; love button ; nice button ; spamming button ; not.great button ; save buttons ; invisible buttons ; improve buttons ; quick buttons ; bigger buttons ; stupid buttons ; little buttons ; great buttons ; many buttons ; small buttons ; nice buttons ; need buttons ; interactive clash ; like clash ; little clash ; nice color ; different colors ; easy colors ; bright colors ; customizable colors ; like colors ; easy crash ; hope crash ; occasional crash ; fixed crash ; frequently crash ; ok crash ; immediately crash ; odd crash

; great crash ; free crash ; crash crash ; excellent crash ; right crash ; forever crash ; latest crashes ; continually crashes ; occasional crashes ; crashes crashes ; works crashes ; hate crashes ; wish crashes ; new crashes ; annoying crashes ; invaluable crashes ; rare crashes ; many crashes ; love crashes ; nice crashes ; lose crashes ; work crashes ; never.easy crashes ; uninstalled crashes ; fewer crashes ; right crashes ; need crashes ; appreciate creaks ; not.unfit crunch ; great crunch ; awesome crunches ; harmful crunches ; tough crunches ; many crunches ; great dark ; good demo ; helpful demo ; great demo ; short demo ; free demo ; excellent demo ; good demos ; great demos ; love demos ; chronologically display ; primary display ; improve display ; better display ; nowhere display ; easy display ; clear display ; straight display ; not.correctly display ; cool display ; properly display ; like display ; large display ; needs display ; not.ready display ; not.customisable display ; keep display ; big display ; great display ; real-time display ; love display ; nice display ; poor display ; perfect display ; minimal displays ; like displays ; good displays ; optimized displays ; first displays ; need displays ; hazardous flag ; allow flag ; stupid font ; larger font ; tiny fonts ; bigger fonts ; small fonts ; popular grumble ; easy gui ; clear gui ; friendly gui ; improved gui ; smooth gui ; new gui ; nice gui ; old icon ; help icon ; same icon ; not.welcoming icon ; little icon ; missing icon ; different icons ; not.understandable icons ; new icons ; fantastic interface ; graphic interface ; lovely interface ; flawless interface ; unintuitive interface ; straightforward interface ; okay interface ; loving interface ; better interface ; solid interface ; n't.great interface ; working interface ; awesome interface ; easy interface ; stupidest interface ; clear interface ; clean interface ; keeps interface ; straight interface ; confusing interface ; dated interface ; smart interface ; unable interface ; old interface ; ok interface ; friendly interface ; high interface ; decent interface ; usable interface ; works interface ; challenging interface ; minimalistic interface ; brilliant interface ; useless interface ; improved interface ; familiar interface ; no.spamming interface ; n't.satisfying interface ; n't.easy interface ; ugly interface ; relevant interface ; like interface ; simpler interface ; selects interface ; smooth interface ; hate interface ; top interface ; good interface ; frustrating interface ; not.keen interface ; streamlined interface ; well interface ; needs interface ; cumbersome interface ; no.intuitive interface ; not.pleasing interface ; slow interface ; allows interface ; new interface ; boring interface ; accurate interface ; super interface ; difficult interface ; amazing interface ; elegant interface ; rubbish interface ; bad interface ; awful interface ; weird interface ; sleek interface ; clumsy interface ; buggy interface ; sucks interface ; not.intuitive interface ; simple interface ; handy interface ; slick interface ; powerful interface ; wrong interface ; great interface ; useful interface ; eye-catching interface ; worthless interface ; awkward interface ; small interface ; best interface ; n't.intuitive interface ; not.friendly interface ; love interface ; nice interface ; efficient interface ; n't.smart interface ; chunky interface ; fits interface ; nicer interface ; poor interface ; excellent interface ; prefer interface ; complicated interface ; intuitive interface ; not.great interface ; average interface ; simplistic interface ; clunky interface ; different interfaces ; nicely interfaces ; new

interfaces ; inconsistencies interfaces ; prettier interfaces ; terrible layout ; busy layout ; different layout ; inconsistent layout ; effective layout ; helps layout ; inconvenient layout ; easy layout ; clear layout ; clean layout ; confusing layout ; not.like layout ; old layout ; beautiful layout ; general layout ; screwed layout ; comfortable layout ; cool layout ; works layout ; appealing layout ; improved layout ; superior layout ; like layout ; good layout ; needs layout ; allows layout ; new layout ; accurate layout ; super layout ; amazing layout ; sucks layout ; simple layout ; great layout ; thoughtful layout ; cheap layout ; love layout ; nice layout ; neat layout ; intuitive layout ; accessible layout ; no.logical layout ; jump light ; bright light ; little light ; dislike light ; new light ; simple light ; odd light ; stuck lights ; wish material ; enough materials ; lose materials ; different message ; better message ; working message ; keeps message ; unable message ; dreadful message ; constant message ; same message ; stupid message ; little message ; boring message ; annoying message ; odd message ; not.fail message ; pops message ; n't.complete message ; crashes messages ; fails messages ; direct messages ; fake messages ; irritating messages ; need messages ; annoying noise ; love noise ; bloated noise ; want noises ; terrifying noises ; constant noises ; creepy noises ; stupid noises ; n't.scary noises ; keeps notification ; customisable notification ; ok notification ; constant notification ; stupid notification ; like notification ; ready notification ; good notification ; new notification ; saver notification ; not.pops notification ; sticky notification ; love notification ; first notification ; need notification ; want notifications ; sound notifications ; kudos notifications ; consistent notifications ; friendly notifications ; help notifications ; spammy notifications ; like notifications ; motivational notifications ; good notifications ; needs notifications ; motivated notifications ; great notifications ; enough notifications ; spam notifications ; love notifications ; nice notifications ; empty notifications ; enjoy notifications ; need notifications ; horrible orange ; meaningless perspective ; correct perspective ; good perspective ; wrong perspective ; great perspective ; uplifting perspective ; n't.expensive perspective ; extra perspective ; top pomp ; stupid popup ; annoying popup ; not.like popups ; force popups ; terrible presentation ; save presentation ; better presentation ; ok presentation ; improved presentation ; good presentation ; new presentation ; great presentation ; nice presentation ; neat presentation ; excellent presentation ; intuitive presentation ; impossible screen ; terrible screen ; want screen ; save screen ; sensitive screen ; pro screen ; easy screen ; n't.irritating screen ; confusing screen ; blocking screen ; bright screen ; crashes screen ; bigger screen ; wide screen ; same screen ; welcoming screen ; customizable screen ; n't.customised screen ; good screen ; keeping screen ; current screen ; new screen ; crashing screen ; ridiculous screen ; keep screen ; stuck screen ; big screen ; saving screen ; crash screen ; fine screen ; small screen ; automatic screen ; love screen ; nice screen ; dedicated screen ; active screen ; poor screen ; n't.keep screen ; first screen ; right screen ; allow screen ; need screen ; different screens ; endless screens ; useless screens ; customizable screens ; many screens ; larger screens ; inconvenient scroll ; easy scroll ; able scroll ; force scroll ; annoying scroll ; awkward scroll ; excellent scroll

; tricky scroll ; impossible search ; n't.want search ; better search ; able search ; quick search ; useless search ; easier search ; helpful search ; recommend search ; need search ; actively seeking ; pro setting ; easy setting ; no.obvious setting ; crashed setting ; uninstalling setting ; missing setting ; good setting ; hard setting ; needs setting ; highest setting ; super setting ; care setting ; helpful setting ; nothing setting ; ridiculous setting ; sucks setting ; great setting ; difficulty setting ; no.problem setting ; love setting ; perfect setting ; first setting ; allow setting ; need setting ; n't.save settings ; terrible settings ; different settings ; sound settings ; save settings ; frustrated settings ; n't.want settings ; not.save settings ; easy settings ; clear settings ; keeps settings ; not.allow settings ; matter settings ; adjustable settings ; same settings ; unnecessary settings ; limited settings ; sluggish settings ; correct settings ; logical settings ; well-thought settings ; new settings ; not.enough settings ; increase settings ; great settings ; n't.retrospective settings ; many settings ; love settings ; nice settings ; lose settings ; strict settings ; perfect settings ; complicated settings ; not.customizable settings ; right settings ; large size ; good size ; smaller size ; increase size ; big size ; no.able size ; small size ; small sizes ; n't.annoying sound ; different sound ; inconsistent sound ; helps sound ; low sound ; alternative sound ; quiet sound ; not.significant sound ; properly sound ; n't.lower sound ; lower sound ; little sound ; good sound ; hard sound ; bad sound ; weird sound ; annoying sound ; great sound ; no.great sound ; nice sound ; superb sound ; fantastic sounds ; terrible sounds ; awesome sounds ; worst sounds ; customizable sounds ; personalize sounds ; like sounds ; needs sounds ; amazing sounds ; weird sounds ; ridiculous sounds ; annoying sounds ; great sounds ; love sounds ; need sounds ; annoying splash ; clever stuff ; terrible stuff ; premium stuff ; awesome stuff ; easy stuff ; important stuff ; keeps stuff ; fixed stuff ; deleting stuff ; stunning stuff ; worst stuff ; cool stuff ; brilliant stuff ; useless stuff ; same stuff ; competitive stuff ; unnecessary stuff ; no.stupid stuff ; regular stuff ; hate stuff ; top stuff ; good stuff ; broken stuff ; new stuff ; tight stuff ; finest stuff ; authentic stuff ; risky stuff ; no.unnecessary stuff ; amazing stuff ; helpful stuff ; awful stuff ; n't.lose stuff ; matters stuff ; simple stuff ; n't.simple stuff ; fancy stuff ; addictive stuff ; great stuff ; loose stuff ; useful stuff ; free stuff ; many stuff ; small stuff ; best stuff ; extra stuff ; cheap stuff ; nice stuff ; crazy stuff ; no.useless stuff ; bare stuff ; expensive stuff ; excellent stuff ; not.interested stuff ; colorful stuff ; n't.extra stuff ; broad sunlight ; common symbols ; fantastic ui ; terrible ui ; unintuitive ui ; convenient ui ; pretty ui ; loving ui ; better ui ; clearer ui ; solid ui ; outdated ui ; easy ui ; clean ui ; keeps ui ; confusing ui ; confusion ui ; n't.good ui ; pleasure ui ; old ui ; beautiful ui ; wonky ui ; friendly ui ; decent ui ; usable ui ; works ui ; improved ui ; same ui ; ugly ui ; like ui ; simpler ui ; compromised ui ; sluggish ui ; fiddly ui ; good ui ; needs ui ; cumbersome ui ; easier ui ; awesome ui ; slow ui ; new ui ; optimized ui ; difficult ui ; amazing ui ; exemplary ui ; smoother ui ; bad ui ; awful ui ; pleasing ui ; clumsy ui ; keep ui ; simple ui ; stuck ui ; slick ui ; shiny ui ; uncluttered ui ; great ui ; free ui ; awkward ui ; spam ui ; love ui ; nice ui ; superb ui ; understandable ui ; neat ui ; poor ui ; excellent ui ; perfect ui ; concerned ui ; complicated

ui ; prettier ui ; intuitive ui ; desirable ui ; rich ui ; clunky ui ; impossible view ; want view ; cos view ; chronological view ; better view ; virtual view ; awesome view ; easy view ; clear view ; keeps view ; able view ; quick view ; unable view ; shocking view ; quickly view ; detailed view ; like view ; ready view ; good view ; hard view ; needs view ; allows view ; new view ; great view ; n't.allow view ; love view ; nice view ; nicer view ; not.able view ; public view ; not.easy view ; need view ; different views ; awesome views ; detailed views ; good views ; new views ; love views ; motivating voice ; n't.annoying voice ; sound voice ; easy voice ; clear voice ; keeps voice ; generic voice ; inspiring voice ; decent voice ; works voice ; freaks voice ; same voice ; n't.little voice ; personalize voice ; hate voice ; little voice ; sexy voice ; good voice ; not.helps voice ; motivates voice ; allows voice ; new voice ; pleasant voice ; dull voice ; awful voice ; newer voice ; weird voice ; n't.work voice ; sweet voice ; annoying voice ; great voice ; valuable voice ; useful voice ; happier voice ; many voice ; awkward voice ; liked voice ; love voice ; nice voice ; excellent voice ; perfect voice ; intuitive voice ; sound voices ; easy voices ; quiet voices ; anoying voices ; many voices ; fake voices ; fast voices ; old white ;

Table 45 The answer to Question 2 from dataset three

exciting activities ; useful activities ; want background ; thank boom ; keep boom ; good boom ; freezes button ; light button ; hate button ; cramped buttons ; okay buttons ; sensitive buttons ; not.big buttons ; small buttons ; best buttons ; like buttons ; good buttons ; good color ; different colors ; awesome colors ; good colour ; interesting crash ; able crash ; not.ok crash ; fun crash ; great crunch ; good display ; great flag ; love flag ; n't.spam flags ; many flags ; light green ; friendly gui ; free icon ; small icons ; understand interface ; better interface ; easy interface ; simple interface ; great interface ; horrible interface ; unique interface ; friendly interface ; nice interface ; improved interface ; good interface ; fantastic layout ; new layout ; hoping layout ; easy layout ; clear layout ; simple layout ; great layout ; nice layout ; perfect layout ; good layout ; jump light ; n't.like light ; breaking light ; not.jump light ; little light ; good light ; broken lights ; terrible lights ; awful lights ; imaginable lights ; ignore lights ; 1st lights ; realistic lights ; good lights ; need lights ; helpful material ; valid material ; great material ; best material ; extra material ; wide material ; same material ; excellent material ; relevant material ; good material ; n't.need materials ; useful materials ; necessary materials ; n't.buy materials ; no.good message ; little message ; clear noise ; keeps noise ; weird noises ; satisfying noises ; useful notification ; not.impressed notifications ; many notifications ; good perspective ; good presentation ; right red ; graphic screen ; slow screen ; different screen ; helps screen ; blocks screen ; working screen ; clear screen ; keeps screen ; stuck screen ; big screen ; freezes screen ; small screen ; ok screen ; bigger screen ; larger screen ; first screen ; good screen ; occasional search ; not.able search ; need search ; graphic setting ; low setting ; immediately setting ; like setting ; low settings ; difficult settings ; improve size ; difficult

size ; smaller size ; increase size ; big size ; not.sufficient size ; small size ; bigger size ;
increasing size ; large size ; good size ; needs size ; bad sound ; not.bad sound ; love sound
; crazy sound ; stupid sound ; good sound ; need sound ; different sounds ; wierd sounds ;
okay sounds ; bad sounds ; easy sounds ; keep sounds ; annoying sounds ; like sounds ;
good sounds ; need sounds ; good splattering ; not.buy stuff ; want stuff ; new stuff ; dumb
stuff ; boring stuff ; helps stuff ; amazing stuff ; helpful stuff ; vital stuff ; luv stuff ;
awesome stuff ; wrong stuff ; great stuff ; free stuff ; extra stuff ; nice stuff ; decent stuff ;
forget stuff ; stupid stuff ; right stuff ; good stuff ; needs stuff ; smoother ui ; brilliant ui ;
needs ui ; different view ; not.enough view ; restricts view ; no.first view ; improve view ;
better view ; not.suitable view ; blocks view ; weird view ; awesome view ; not.good view
; clear view ; sucks view ; unable view ; not.needs view ; extra view ; negative view ; nice
view ; block view ; challenging view ; larger view ; moving view ; perfect view ; realistic
view ; hazardous view ; first view ; missing view ; good view ; different views ; not.good
views ; need views ; helpful voice ; bad voice ;

For other questions in that question list, Appendix G Answers to the SPARQL queries about the applications of the GMAURO ontology provides the corresponding Snap SPARQL Queries and the screenshots from the answers in dataset one.

5 Discussion

This chapter discusses various topics that researchers may find interesting after reading this thesis. Section 5.1 summarizes this thesis. Section 5.2 discusses the adaptability of this prototype in the measures and a kind of potential “contrary linguistic rule” in the future. Section 5.3 reviews the advantages and disadvantages of this framework and each component. Section 5.4 reflects the flexibility of this framework on both component level and logics and algorithms level. Section 5.5 discusses the performance impact of a minor bug appearing in the topic-opinion extractor traversal function version two. Section 5.6 suggests some future research directions of this framework.

5.1 Summary of this thesis

The aim of this PhD is to develop a framework that reads user review data and extracts meaningful information to support stakeholders in improving their products by satisfying user requirements. This PhD meets the four objectives that are outlined at the beginning of this research:

- (1) To automatically extract app reviews from a major app store platform.

The user review web sourcing component provides a flexible method that extracts user reviews from Google Play and that could evolve with the webpage changes along with time.

- (2) To analyse the review data in order to produce meaningful information.

The implemented prototype of the proposed framework analyses the user reviews with manually programmed linguistic rules.

- (3) To research and develop a method to evaluate the produced information.

The produced information is evaluated in both random samples from the databases and the General Mobile App User Review Ontology.

- (4) To validate the framework using examples from two or three mobile app user reviews datasets.

Three mobile app user review datasets from diabetes management, fitness training management and driving theory were used to evaluate the functionalities.

5.2 Adaptability of this prototype

5.2.1 Measures

This prototype has two components that use linguistic rules. Both sets of linguistic rules are adaptable.

The way to adapt the user request elicitor component to new datasets is introduced in the 3.4 section. The way to adapt the topic-opinion extractor component has some more information explained below.

The topic-opinion extractor component can be adapted to new datasets in three aspects: popular opinions adaptation, keywords list adaptation, and the linguistic rules adjustments.

Popular opinions can be adapted through the first version of the component, which produces a table of topic and mapping opinions. Through this table, the adaptation is easier than doing by hand. In practice, future researchers can rule out other words that are not nouns, adjectives, verbs, and adverbs. A selection that takes into account the use scenarios will help to decide which opinions are really helpful in the dataset.

A piece of code that helped in the popular opinions adaptation from version one to version two is presented in the List of Code as C.17 Python code to help the adaptation of popular opinions. To use this piece of code, users need to update the popular opinions list with their own version that is in use, and convert the opinions, which exist in the pairsTable in the database from running version one, into the “opinions” list in the C.17 code.

It is necessary to remind users of this prototype that adaptation of popular opinions could mean adding opinions and removing some unsuitable opinions for the new datasets and domains.

Keywords adaptation also means adding keywords to the list or removing some unsuitable keywords from the list. But the process is simpler. Users are recommended to select the topics from the “subject” field from the pairsTable from running version one and group by the

“subject”, thereby it is possible to order by the frequencies of them. Then the resulting list will be helpful for the keyword adjustments for new datasets and domains.

Pairing linguistic rules’ adjustments are through the rules array in the debugging code C.13 and C.14. Once the adjustments are completed, they need to be reflected into the code that talks to the database, C.9 and C.10.

However, the linguistic rules introduced previously for this component is for the purpose of allowing the topic and opinion pairs to appear in the results. As a feature of version two, another purpose that disallows the appearance becomes possible through a set of “contrary linguistic rules”.

The next subsection explains this measure in detail by potentially introducing a “contrary linguistic rule” concept in the version two of the Topic-Opinion extractor in the future.

5.2.2 Contrary linguistic rules

A “contrary linguistic rule” is a potential new concept that is possible to implement in version two of the Topic-Opinion extractor. Sometimes an opinion is a polysemous word. When the word appears as a real opinion, it is exactly what is needed. However, when the word appears in other meanings other than an opinion, if there are no other linguistic rules to bind it at this time, it will become an orphan opinion, which will reduce the system's performance.

The purpose for contrary linguistic rules is to bind such unwanted words with a contrary rule. The rule is strange enough therefore guarantees that the unwanted word could not pass the keyword test, in order to prevent it from appearing as an orphan opinion.

Some examples of such contrary rules are below:

[182, 'mwe', 'RB', 'RB', 'parent'],

[183, 'case', 'IN', 'RB', 'child'],

[184, 'case', 'IN', 'NN', 'child'],

[185, 'mark', 'IN', 'VBP', 'child'],

[186, 'case', 'IN', 'NNS', 'child'],

[187, 'case', 'IN', 'JJ', 'child'],

[188, 'mark', 'IN', 'VBD', 'child'],

[189, 'mark', 'IN', 'VBG', 'child']

Taking rule 184 as an example, this rule can catch scenarios like:

"they look *like* string art !" (1)

and

"photo at *top* of charity selection list loads , but no list of charities below it ." (2)

Figure 32 illustrates how these above two sentences are parsed in their dependency trees. Both “like” and “top” are in the popular opinions list, but they appear as an adverb “like” and a noun “top” in these two examples that do not carry the meanings of the specific opinions. If they are extracted as orphan opinions, the resulting data would not be what they are supposed to be there. For example, the sentence (1) produces an orphan opinion “like” that links to “app”, and sentence (2) produces an orphan opinion “top” that links to “app”. Both are false positive results. In order to overcome this problem, contrary linguistic rules are used. Rule 184 works in both these two sentences. Therefore, they can form a relationship with another word, but the relationship pair will not pass the keyword test.

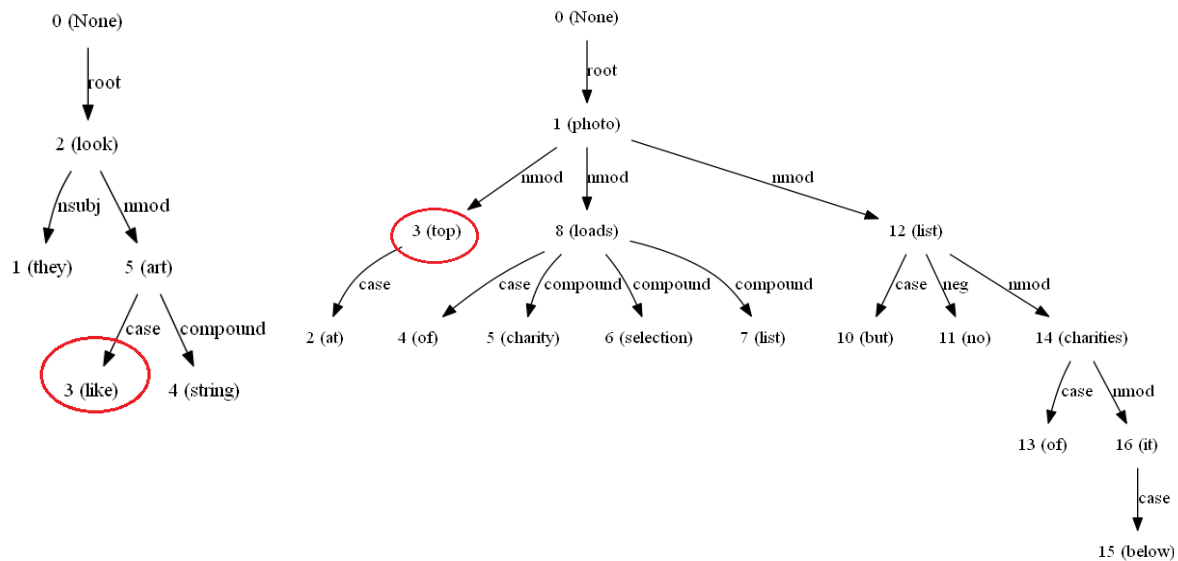


Figure 32 Examples of contrary linguistic rules' usage

5.3 Advantages and disadvantages of each component and the framework

This prototype has five components. The advantages and disadvantages of each component are introduced as follows.

1) The set of web sourcing code

The advantage of the set of web sourcing code is performance. From the information retrieval point of view, as long as it passes the proper unit testing, 100% of the target information will be retrieved.

Its disadvantage is that it is easy to break. This is because Google frequently updates the website. The updated content may be a simple tag attribute update, or a large web page structure change, or even an element may be obscured by another element. Therefore, maintaining this set of code according to the changes in the website is challenging.

2) SO-CAL

The advantage of SO-CAL is that it can calculate a sentiment score for each review. The sentiment polarities generated by SO-CAL's sentiment calculations are relatively accurate.

Its disadvantage is that it needs to be adjusted to target the language characteristics of the new application domains to achieve better accuracy in sentiment score calculation. In this component, this is the work that has not yet been completed.

3) The user request elicitor

The first advantage of this component is that the precision performance can be controlled according to the rules. Moreover, each rule can be flexible, in terms of precision trade-offs, in terms of the scope of the included scenarios, and in the structural aspects of the rules themselves. In addition, it is entirely possible to add new rules and even completely replace existing rules with another set of rules to solve another research question.

Its disadvantage is that it does not use advanced NLP techniques such as entity recognition, semantic, and co-reference. In terms of information granularity, it is not detailed enough. The extracted user requests are currently only in the form of a single piece of information. At the same time, the maintenance challenges posed by the grammar of linguistic rules also exist.

4) The Topic-Opinion Extractor

The first advantage of this component is that it is a new way to use the Stanford dependency tree. It solves an old problem relatively accurately with a new perspective. The matching method of topics and opinions before this component is not precise. The performance of this component, as reported in the Results and evaluation chapter, is relatively good, and close to the performance limits that can be achieved under existing structures. Moreover, due to the use of Open Closed Principle, the maintenance of rules is quite easy. In addition, it also provides the possibility of easily adding new rules, even replacing existing rules with a new set of rules.

The downside is that Stanford Parser is called once for every sentence, which slows down the code. Stanford Parser is likely to be replaced by Stanford CoreNLP in the future, and this issue is likely to be resolved by Stanford CoreNLP.

5) GMAURO and the ontology populator

This component has three advantages. The first is that it accomplishes automatic topic classification. This is achieved by the correspondence between the keywords list and the concepts under the "Subject" class of the ontology. The second is that GMAURO is probably the first ontology in the mobile app user review area. The third is that this component uses a potentially new algorithm to accomplish the population of a large number of different types of individual data and complex relationships.

Its disadvantage is that it does not use advanced semantic matching techniques, but simply uses keyword filtering to complete the concept mapping.

6) Advantages and disadvantages of the framework

There are some differences between this framework and conventional computational linguistic artefacts. This framework is able to form an architecture that links a series of components to fulfil user review analysis. It could take more costs and resources to develop a similar scale AI system.

Compared with an AI system, this framework is easier to understand in logics and algorithms. All data are clearly traceable in the data flow. This facilitates the debugging and future

development for stakeholders and other researchers. This framework does not require a large scale of labelled data during the development, nor in the adaptation to new environments.

However, this framework has to rely on data analysis up to a satisfactory amount in the linguistic rules' identification. As shown in the results, user requests and pairs of topics and opinions are returned with good precision, but lower recall. This is because people express requests and opinions in so many ways, which are also constantly evolving over the time.

Moreover, the NLP techniques used in this framework are fundamental. This limits the functionalities and information granules in many ways. For example, if co-references can be analysed among different sentences in a review, information mining would be more precise. The current Topic-Opinion extractor works against a single dependency tree each time. Therefore, it awkwardly deals with scenarios in such a complex user review:

“Wish I never updated. This version is useless to me. The old version was perfect as it was. I used it for years & it was so easy use. Even my Dr liked it. Sadly, this is so awful, I will be looking for another app.”

5.4 Flexibility of this framework

5.4.1 Flexibility in component level

This framework is flexible in the ways it is built and of its future development. All components are standalone, and also relate to each other. None of two components are hardcoded together. Therefore, updates on any component can take place without demanding changes from others. Depending on the scope of the change in logic, updates can also perform between some of the components. The reasons for the way that they are currently structured have been explained briefly in previous sections of the components. In the meantime, stakeholders do have freedom to change the structure, or replace anyone of the components. These could happen in the forms as follows, but not limited by them.

- Update the web sourcing code to new versions towards Google Play.
- Replace the web sourcing code to source reviews from other data sources.

- Update the logic between Request Elicitor and Topic-Opinion Extractor to become parallel, though not recommended.
- Add more components in the structure as parallels with more analysis accomplished before populating into the ontology.
- Update the GMAURO ontology with concepts that interest stakeholders more in terms of specific analysis.
- Replace GMAURO with another ontology that is designed for another domain, then this framework can expand to other domains.
- Add other ontologies that can distinguish terminologies related to specific domains, then the analysis could either rule out the influences by domain terminologies, or be improved in information granules in those domains.

5.4.2 Flexibility in logics and algorithms level

This framework also has flexibilities in logics and algorithms level to improve functionalities. For example, there are at least several such directions for further development.

The first direction is further sentiment analysis against the opinions, such as allocating a score for each opinion, which will improve the resulting picture on quantitative analysis on sentiments in line with topics.

The second direction is to apply a similar subject category analysis mechanism to the elicited user requests. Therefore, the new user requirements in the requests can be decomposed into subject topic level and classified into categories.

The third direction can be replacing the current linguistic rules with a set of new rules for new research questions. For example, replacing the user request rules with a set of user complaint rules, or replacing the topic-opinion rules with a set of adverb-opinion rules to extract how strong the opinions are.

The fourth direction will be to further extend the processing of the dependency tree, extending the single word topics and opinions into phrases, thus gaining context support.

5.5 A minor bug in the topic-opinion extractor traversal function version two

There is a minor bug hidden in the code C.10 in this submission. It will be corrected after the submission and the performance will be re-evaluated to confirm that there is no big impact. However, the reason for this bug is reported here and the potential performance impact is also evaluated.

In the topic-opinion extractor traversal function version two, the current nodes are checked against the popular opinions list before the relationships that involve the current nodes are returned. This is to ensure that all resulting opinions are popular opinions in version two.

However, in the current code, there are possibilities for terms that are not popular opinions to return in relationships as the opinions. The code snippet below shows the possibilities:

Code Snippet 19 The code relates to this bug

```
for rule in rules:
    if rule[1] == rel and rule[2]==childTag and rule[3]==parentTag:
        if rule[4]=='parent' and goldenOpinionFlag == True:
            related_pairs.append((headNumber, originalNode['address'], neg, rule[0]))
            ruleMatchedFlag = True
            print("inserted according to rule "+str(rule[0])+": "+str(headNumber)+",
"+str(originalNode['address'])+" and "+str(neg))
            elif rule[4]=='child' and goldenOpinionFlag == True:
                related_pairs.append((originalNode['address'], headNumber, neg, rule[0]))
                ruleMatchedFlag = True
                print("inserted according to rule "+str(rule[0])+": "+str(originalNode['address'])+",
 "+str(headNumber)+" and "+str(neg))
```

The code in black lines is the right code. The code in grey and italic fonts introduces this bug. This bug allows a relationship that has the current node as the subject and the parent of the current node as the opinion returns. The True value of the flag is stating that the current node is a popular opinion. But no test is done for what its parent node is. Its parent could be a popular opinion, an opinion but not popular, or noise.

All relationships then have to undertake the keywords list check for their topics. When this bug happens, the current nodes that are popular opinions become the topics. Therefore, such relationships could only pass the keywords list filtering and eventually enter the final results when the current nodes are words that are in the intersection set of popular opinions list and the keywords list. These words are 13 in total in version two at present:

“detailed, help, helped, informative, motivation, motivator, work, works, graphic, light, sound, competition, crashes”

The number of final relationships that have these subjects can be counted by this SQL query:

```
select count(*) from pairstable where subject = "crashes" or subject = "competition" or
subject = "sound" or subject = "light" or subject ="graphic" or subject ="detailed" or
subject = "help" or subject = "helped" or subject = "informative" or subject = "motivation"
or subject ="motivator" or subject = "work" or subject ="works";
```

It is not clear how many of these records have passed the traversal function through the branch where this bug is. Roughly this can be assumed as 50%.

Among these 50% of the records, because they are actually checked by specific linguistic rules, the precision would be that of the linguistic rules, which on a modest estimation should be 80% on average. Therefore, the noise data should be less than or approximately 20%.

Then the estimation of the percentages of the noise data should be:

$$\text{Percentages of the 13 subjects in final records} * 50\% * 20\% ,$$

which are calculated below:

Table 46 Calculations of the negative impact of this bug on precisions

	Dataset one	Dataset two	Dataset three
Final records containing the 13 subjects	843	3333	526
Total final relationships in the datasets	39783	124426	60111
Percentages of the 13 subjects in final records	2.119%	2.6787%	0.875%

The estimated resulting noise data percentages	0.2119%	0.26787%	0.0875%
--	---------	----------	---------

Table 46 shows that this bug introduces negative impact on precisions for the three datasets: 0.2119%, 0.26787% and 0.0875% respectively.

Because there are possibilities that this bug could bring more records into the final results, the impact on recalls would be positive. However, the actual values should be very trivial and safe to ignore. Among those 13 subjects related records, if the opinions are popular opinions, they will go through the correct branches eventually, therefore the contributions of this bug to recalls are only from the opinions that are not popular. It is worth noting that these non-popular opinions do not include unqualified noise data because noises cannot be counted into the calculation of recall. These non-popular opinions' percentage is small, which could be assumed to be around 5%. Moreover, it has to be based on the percentages of the 13 subjects in final records also. Furthermore, the possibility of such non-popular opinions going through the branches where the bug is can also be assumed to 50%. The formula that estimates such contributions should be:

$$\text{Percentages of the 13 subjects in final records} * 50\% * 5\%$$

The resulting contributions will be much less than the impacts on precisions, therefore it is safe to ignore the impact on recalls.

Such a bug that has such a trivial impact on performance is challenging for black box software testing that does not inspect internal logics.

Updates on the fix and testing of this minor bug:

The fix of this minor bug has two steps:

- (1) When doing the rule tests between the current node and its parent, remove the second branch. This is to remove the code in grey colour in Code Snippet 19.
- (2) When doing the rule tests between the current node and its children, remove the first branch.

Testing of this minor bug:

Test case (being affected by this bug):

“I have sent feedback reports with several crashes.”

The results in the pairstable (before the keyword comparison step) before this fix:

id	reviewId	sentenceId	subject	negation	opinion	ruleNumber
1859	306	5	crashes	sent		77
1860	306	5	crashes	several		105

Both above two resulting records decrease the precision in the performance. The first line of records goes with the second branch (child is the subject) of the rule tests between the current node and its parent. In this example, when “crashes” is the current node and it’s a popular opinion, where its parent “sent” isn’t a popular opinion, this pair matches rule 77 and entered the results, therefore decreasing the precision. The second line of records goes with the first branch (parent is the subject) of the rule tests between the current node and its children. In this example, when “crashes” is the current node and it’s a popular opinion, where its child “several” is not a popular opinion, this pair matches rule 105 and entered the results, therefore also decreasing the precision.

After this fix, both those two records disappeared from the results.

Another test case that was not affected by this bug is also tested after this fix:

“I really love this cross platform diary that syncs effortlessly between my Android and Windows 10 devices.”

The results that produced to pairstable remain unchanged:

id	reviewId	sentenceId	subject	negation	opinion	ruleNumber
1	1	1	diary	love		38
2	1	1	app	effortlessly		1000

Therefore, it should be reasonable to conclude that this fix works and there is no knock down effect found in this unaffected test case.

Performance confirmation:

Because this minor bug is calculated and predicted with very subtle effect to the results, the performance is difficult to test specifically on a large scale. However, a performance confirmation is carried out together with the next version of code, and it will be either reported at a later time, or submitted to Brookes RADAR.

5.6 Future research directions

Fully developing the flexibility of this framework would be the most interesting research direction in the future at the component level and logic algorithm level. These have been introduced in the previous discussion section 5.4 on flexibilities.

Short-term research directions are suggested for further research on contrary linguistic rules that will enhance the performance, the rules for the strength of opinions, extending the user's request elicitation to the extraction of complaints.

The research that can be conducted in the future may also be to refine the granularity of information and to obtain contexts, such as decomposing user requests or complaints obtained, refining sentiment analysis of resulting opinions, and extending the processing of dependency trees.

For example, decomposing the user requests or complaints obtained is a direction to refine the granularity of information. There are three issues in this direction. Firstly, decomposing the single pieces of information of the resulting requests or complaints (or any other information that the future researchers have elicited according to their own rules) is necessary. To solve this issue, the Stanford parser can be used here to find the key information that is most interesting. Certain experiments will be needed here to decide which part of the tree should be useful. However, there might be other ways to do that, such as narrowing down the chunks to some key kernel phrases that match certain rule-related POS patterns.

Secondly, in order to merge same entities or similar phrases, certain similarity calculations are necessary here. Therefore, the same individuals can be grouped together, and even similar individuals can form groups or clusters.

Thirdly, both above measures have to be independent of the structures, namely they can deal with any information that is produced from the customized information elicitor components.

Moreover, it will be great if the ontology can answer such questions as “what are the top 10 requirements you have elicited from the user’s reviews?” or “what are the top 50 topics that users were talking about?” In order to answer these two questions, the GMAURO ontology will need counters (and maybe flags) for requests and topics individuals. These counters are responsible to check whether an individual is the first one and if not, how many there are. However, there is a danger for a too simple solution that might not be able to answer a further question such as “what are the top 5 opinions that the users hold against a specific function point?” This type of question needs calculations of the relationships between the types of individuals involved in the question. Therefore, a careful design of the new version of GMAURO in terms of the counters (and maybe flags) on classes and relationships is recommended.

Future researchers are welcome if they aim to work in these above directions, apply this framework to more areas, or change the structure to fit the web service architecture.

6 Conclusion

User review analysis is an important complement to the traditional requirement elicitation methods. In the field of user review analysis, the commonly used technology is machine learning, which has the disadvantage of being difficult to control internal details. In opinion mining, researchers either extract topics and opinions separately and make efforts to bridge between them, or extract them together. But the exact mapping relationships between topics and opinions are rarely produced. User requests are also rarely elicited successfully in practice. An overall solution that tackles the problems from the original user reviews to the resulting user requests and opinion feedback has not been found. Here I show a framework that can cover the entire process from the original user reviews to the final extracted user feedback on new and existing requirements, as well as solve the various problems therein. I found the whole process can be achieved through a few loosely connected components and driven by linguistic rules that are manually programmed. The best performance that a user review analysis system can achieve has a limit, which is determined by the different ways in which users express information, although this limit may vary in different datasets. Furthermore, I found that this non-machine learning approach avoids the typical problems of topic domain dependency and overfitting. My results demonstrate how the performance of the system and the internal logic details can interact and are controlled through the participation of linguistic rules. I anticipate this thesis to be a starting point for a more comprehensive and enhanced user review analysis framework. For example, more questions such as what the user complaints are, how strong the users' opinions are, could be answered through the enrichment or replacement of the linguistic rules. Furthermore, finer granularity of information and phrases level contexts are valuable for information exchange in a broader range of areas, and well-defined further research at the component level or logic algorithm level will be extendable to such developments.

7 References

- [1] M. Arjun, V. Vivek, L. Bing, and G. Natalie, “Fake Review Detection: Classification and Analysis of Real and Pseudo Reviews,” UIC-CS-03-2013, 2013.
- [2] M. Crawford, T. M. Khoshgoftaar, J. D. Prusa, A. N. Richter, and H. Al Najada, “Survey of review spam detection using machine learning techniques,” *Journal of Big Data*, vol. 2:23, 2015.
- [3] Z. You, T. Qian, and B. Liu, “An Attribute Enhanced Domain Adaptive Model for Cold-Start Spam Review Detection,” *Proceedings of the 27th International Conference on Computational Linguistics*, pp. 1884–1895, 2018.
- [4] S. P. Rajamohana, K. Umamaheswari, and S. V. Keerthana, “An effective hybrid Cuckoo Search with Harmony search for review spam detection,” *3rd International Conference on Advances in Electrical, Electronics, Information, Communication and Bio-Informatics, AEEICB 2017*, pp. 524–527, 2017.
- [5] J. Li, M. Ott, C. Cardie, and E. Hovy, “Towards a General Rule for Identifying Deceptive Opinion Spam,” *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1566–1576, 2014.
- [6] Google Play Developer API, “Reviews,” 2017. [Online]. Available: <https://developers.google.com/android-publisher/api-ref/reviews>. [Accessed: 03-Aug-2019].
- [7] App Annie, “App Annie,” 2019. [Online]. Available: <https://www.appannie.com/en/>. [Accessed: 03-Aug-2019].
- [8] AppTopia, “AppTopia,” 2019. [Online]. Available: <https://apptopia.com/>. [Accessed: 03-Aug-2019].
- [9] GOV.UK, “Copyright Acts and related laws,” 2011. [Online]. Available: <https://www.gov.uk/government/publications/copyright-acts-and-related-laws>. [Accessed: 25-Aug-2019].

- [10] EUR-Lex, “GDPR,” 2016. [Online]. Available: <https://eur-lex.europa.eu/eli/reg/2016/679/2016-05-04>. [Accessed: 25-Aug-2019].
- [11] legislation.gov.uk, “Data Protection Act 2018,” 2018. [Online]. Available: http://www.legislation.gov.uk/ukpga/2018/12/contents/enacted?_ga=2.157571450.1590840207.1566662099-1478164772.1523874321. [Accessed: 25-Aug-2019].
- [12] EU GDPR.ORG, “GDPR FAQs,” 2018. [Online]. Available: <https://eugdpr.org/the-regulation/gdpr-faqs/>. [Accessed: 25-Aug-2019].
- [13] S.-M. Kim and E. Hovy, “Determining the sentiment of opinions,” *Proceedings of the 20th international conference on Computational Linguistics*, pp. 1367–1373, 2004.
- [14] Y.-Y. Zhao, B. Qin, and T. Liu, “Sentiment analysis,” *Journal of Software*, vol. 21, no. 8, pp. 1834–1848, 2010.
- [15] M. Wiegand, A. Balahur, B. Roth, D. Klakow, and A. Montoyo, “A Survey on the Role of Negation in Sentiment Analysis,” *Proceedings of the Workshop on Negation and Speculation in Natural Language Processing*, pp. 60–68, 2010.
- [16] K. Ravi and V. Ravi, “A survey on opinion mining and sentiment analysis: Tasks, approaches and applications,” *Knowledge-Based Systems*, vol. 89, pp. 14–46, 2015.
- [17] T. A. Rana and Y. N. Cheah, “Aspect extraction in sentiment analysis: comparative analysis and survey,” *Artificial Intelligence Review*, vol. 46, no. 4, pp. 459–483, 2016.
- [18] K. Schouten and F. Frasincar, “Survey on Aspect-Level Sentiment Analysis,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 3, pp. 813–830, 2016.
- [19] S. Sun, C. Luo, and J. Chen, “A review of natural language processing techniques for opinion mining systems,” *Information Fusion*, vol. 36, pp. 10–25, 2017.
- [20] M. Soleymani, D. Garcia, B. Jou, B. Schuller, S. F. Chang, and M. Pantic, “A survey of multimodal sentiment analysis,” *Image and Vision Computing*, vol. 65, pp. 3–14, 2017.
- [21] N. Genc-Nayebi and A. Abran, “A systematic literature review: Opinion mining studies from mobile app store user reviews,” *Journal of Systems and Software*, vol. 125, pp. 207–219, 2017.

- [22] M. Tavakoli, L. Zhao, A. Heydari, and G. Nenadić, “Extracting useful software development information from mobile application reviews: A survey of intelligent mining techniques and tools,” *Expert Systems with Applications*, vol. 113, pp. 186–199, 2018.
- [23] I. Chaturvedi, E. Cambria, R. E. Welsch, and F. Herrera, “Distinguishing between facts and opinions for sentiment analysis: Survey and challenges,” *Information Fusion*, vol. 44, pp. 65–77, 2018.
- [24] M. Munezero, C. S. Montero, E. Sutinen, and J. Pajunen, “Are they different? affect, feeling, emotion, sentiment, and opinion detection in text,” *IEEE Transactions on Affective Computing*, vol. 5, no. 2, pp. 101–111, 2014.
- [25] V. Hatzivassiloglou and K. R. McKeown, “Predicting the semantic orientation of adjectives,” *35th Annual Meeting of the Association for Computational Linguistics and 8th Conference of the European Chapter of the Association for Computational Linguistics*, pp. 174–181, 1997.
- [26] J. M. Wiebe, “Learning Subjective Adjectives from Corpora,” *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pp. 735–740, 2000.
- [27] P. D. Turney and M. L. Littman, “Measuring Praise and Criticism: Inference of Semantic Orientation from Association,” *ACM Transactions on Information Systems*, vol. 21, no. 4, pp. 315–346, 2003.
- [28] E. Riloff and J. Wiebe, “Learning extraction patterns for subjective expressions,” *Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing*, pp. 105–112, 2003.
- [29] S. Kim and E. Hovy, “Automatic Detection of Opinion Bearing Words and Sentences,” *Proceedings of IJCNLP-05, the Second International Joint Conference on Natural Language Processing*, vol. companion, pp. 61–66, 2005.
- [30] A. Andreevskaia and S. Bergler, “Mining WordNet for fuzzy sentiment: Sentiment tag extraction from WordNet glosses,” *EACL 2006 - 11th Conference of the European Chapter of the Association for Computational Linguistics*, pp. 209–216, 2006.

- [31] F. Su and K. Markert, "Subjectivity recognition on word senses via semi-supervised mincuts," *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 1–9, 2009.
- [32] D. Maynard and A. Funk, "Automatic detection of political opinions in Tweets," *The Semantic Web: ESWC 2011 Workshops.*, pp. 88–99, 2011.
- [33] I. Titov and R. McDonald, "Modeling online reviews with multi-grain topic models," *Proceeding of the 17th international conference on World Wide Web - WWW 2008*, pp. 111–120, 2008.
- [34] V. Stoyanov and C. Cardie, "Topic identification for fine-grained opinion analysis," *Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008)*, pp. 817–824, 2008.
- [35] G. Qiu, B. Liu, J. Bu, and C. Chen, "Opinion word expansion and target extraction through double propagation," *Computational Linguistics*, vol. 37, no. 1, pp. 9–27, 2011.
- [36] C Cardie, V. Stoyanov, Y. Choi, and E. Breck, "System and method for automatically summarizing fine-grained opinions in digital text," US Patent 8,280,885 B2, 2012.
- [37] Y. Kang and L. Zhou, "RubE : Rule-based methods for extracting product features from online consumer reviews," *Information & Management*, vol. 54, no. 2, pp. 166–176, 2017.
- [38] T. A. Rana and Y. N. Cheah, "A two-fold rule-based model for aspect extraction," *Expert Systems with Applications*, vol. 89, pp. 273–285, 2017.
- [39] S. Bethard, H. Yu, A. Thornton, V. Hatzivassiloglou, and D. Jurafsky, "Automatic Extraction of Opinion Propositions and their Holders," *Proceedings of the AAAI Spring Symposium on Exploring Attitude and Affect in Text: Theories and Applications*, 2004.
- [40] Y. Choi, C. Cardie, L. Riloff, and S. Patwardhan, "Identifying sources of opinions with conditional random fields and extraction patterns," *Proceedings of Human*

Language Technology Conference and Conference on Empirical Methods in Natural Language Processing, pp. 355–362, 2005.

- [41] S.-M. Kim and E. Hovy, “Identifying and analyzing judgment opinions,” *HLT-NAACL ’06: Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, pp. 200–207, 2006.
- [42] S.-M. Kim and E. Hovy, “Extracting opinions, opinion holders, and topics expressed in online news media text,” *Proceedings of the Workshop on Sentiment and Subjectivity in Text*, pp. 1–8, 2006.
- [43] M. Wiegand and D. Klakow, “Convolution kernels for opinion holder extraction,” *NAACL HLT 2010 - Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 795–803, 2010.
- [44] A. Gangemi, V. Presutti, and D. Reforgiato Recupero, “Frame-based detection of opinion holders and topics: A model and a tool,” *IEEE Computational Intelligence Magazine*, vol. 9, no. 1, pp. 20–30, 2014.
- [45] J. Yi, T. Nasukawa, R. Bunescu, and W. Niblack, “Sentiment analyzer: extracting sentiments about a given topic using natural language processing techniques,” *Proceeding of ICDM-03, the 3rd IEEE International Conference on Data Mining*, pp. 427–434, 2003.
- [46] K. Bloom, N. Garg, and S. Argamon, “Extracting appraisal expressions,” *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*, pp. 308–315, 2007.
- [47] The Stanford Natural Language Processing Group, “Stanford Parser,” 2018. [Online]. Available: <https://nlp.stanford.edu/software/lex-parser.shtml>. [Accessed: 09-Jan-2019].
- [48] The Stanford Natural Language Processing Group, “The Stanford Parser: A statistical parser,” 2018. [Online]. Available: <https://nlp.stanford.edu/software/lex-parser.shtml>.

[Accessed: 05-Aug-2019].

- [49] M. Hu and B. Liu, “Mining opinion features in customer reviews,” *AAAI’04: Proceedings of the 19th national conference on Artificial intelligence*, pp. 755–760, 2004.
- [50] A.-M. Popescu and O. Etzioni, “Extracting Product Features and Opinions from Reviews,” *HLT ’05: Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pp. 339–346, 2005.
- [51] W. X. Zhao, J. Jiang, H. Yan, and X. Li, “Jointly modeling aspects and opinions with a MaxEnt-LDA hybrid,” *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pp. 56–65, 2010.
- [52] F. Li, S. J. Pan, O. Jin, Q. Yang, and X. Zhu, “Cross-domain co-extraction of sentiment and topic lexicons,” *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 410–419, 2012.
- [53] K. Liu, L. Xu, and J. Zhao, “Extracting opinion targets and opinion words from online reviews with graph co-ranking,” *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 314–324, 2014.
- [54] K. Liu, L. Xu, and J. Zhao, “Co-extracting opinion targets and opinion words from online reviews based on the word alignment model,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 3, pp. 636–650, 2015.
- [55] S. Jebbara and P. Cimiano, “Aspect-based relational Sentiment analysis using a stacked neural network architecture,” *Proceedings of the Twenty-Second European Conference on Artificial Intelligence*, vol. 285, pp. 1123–1131, 2016.
- [56] W. Wang, S. J. Pan, D. Dahlmeier, and X. Xiao, “Coupled multi-layer attentions for co-extraction of aspect and opinion terms,” *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, pp. 3316–3322, 2017.
- [57] J. Yu, J. Jiang, and R. Xia, “Global inference for aspect and opinion terms co-extraction based on multi-task neural networks,” *IEEE/ACM Transactions on Audio*,

Speech and Language Processing (TASLP), vol. 27, no. 1, pp. 168–177, 2019.

- [58] F. Bravo-marquez, E. Frank, and B. Pfahringer, “Annotate-Sample-Average (ASA): A New Distant Supervision Approach for Twitter Sentiment Analysis,” *Proceedings of the Twenty-Second European Conference on Artificial Intelligence*, pp. 498–506, 2016.
- [59] M. Taboada, J. Brooke, M. Tofiloski, K. Voll, and M. Stede, “Lexicon-Based Methods for Sentiment Analysis,” *Computational Linguistics*, vol. 37, no. 2, pp. 267–307, 2011.
- [60] O. Kolchyna, T. T. P. Souza, P. Treleaven, and T. Aste, “Twitter Sentiment Analysis: Lexicon Method, Machine Learning Method and Their Combination,” in *Handbook of Sentiment Analysis in Finance*, 2016.
- [61] C. Dhaoui, C. M. Webster, and L. P. Tan, “Social media sentiment analysis: lexicon versus machine learning,” *Journal of Consumer Marketing*, vol. 34, no. 6, pp. 480–488, 2017.
- [62] W. Medhat, A. Hassan, and H. Korashy, “Sentiment analysis algorithms and applications: A survey,” *Ain Shams Engineering Journal*, vol. 5, no. 4, pp. 1093–1113, 2014.
- [63] J. Polpinij and A. K. Ghose, “An ontology-based sentiment classification methodology for online consumer reviews,” *Proceedings of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology - Volume 01*, pp. 518–524, 2008.
- [64] M. Taboada, “SO-CAL,” 2018. [Online]. Available: <https://github.com/sfu-discourse-lab/SO-CAL>. [Accessed: 28-Aug-2019].
- [65] A. C. Cosma, V. V. Itu, D. A. Suciu, M. Dinsoreanu, and R. Potolea, “Overcoming the domain barrier in opinion extraction,” *2014 IEEE 10th International Conference on Intelligent Computer Communication and Processing (ICCP)*, pp. 289–296, 2014.
- [66] NLTK Project, “Installing NLTK,” 2019. [Online]. Available: <https://www.nltk.org/install.html>. [Accessed: 15-Dec-2019].
- [67] M. Taboada, “SO-CAL Project Homepage,” 2011. [Online]. Available:

<https://www2.cs.sfu.ca/~sentimen/social/>. [Accessed: 12-Aug-2019].

- [68] Princeton University, “WordNet,” 2019. [Online]. Available: <https://wordnet.princeton.edu/>. [Accessed: 12-Aug-2019].
- [69] C. Burnay, I. J. Jureta, and S. Faulkner, “Context-Driven Elicitation of Default Requirements: an Empirical Validation,” arXiv:1211.2620, 2012.
- [70] J. C. S. Do Prado Leite and A. P. M. Franco, “A strategy for conceptual model acquisition,” [1993] *Proceedings of the IEEE International Symposium on Requirements Engineering*, pp. 243–246, 1993.
- [71] The Object Management Group, “Semantics Of Business Vocabulary And Rules,” 2019. [Online]. Available: <https://www.omg.org/spec/SBVR/About-SBVR/>. [Accessed: 02-Dec-2019].
- [72] S. Kujala, M. Kauppinen, L. Lehtola, and T. Kojo, “The role of user involvement in requirements quality and project success,” *Proceedings of the 13th IEEE International Conference on Requirements Engineering*, pp. 75–84, 2005.
- [73] N. Seyff, I. Todoran, K. Caluser, L. Singer, and M. Glinz, “Using popular social network sites to support requirements elicitation, prioritization and negotiation,” *Journal of Internet Services and Applications*, vol. 6, 2015.
- [74] M. Harrod, “Chunking,” 2019. [Online]. Available: <https://www.interaction-design.org/literature/book/the-glossary-of-human-computer-interaction/chunking>. [Accessed: 26-Nov-2019].
- [75] S. Levithan, “Regex Legends: The People Behind the Magic,” 2008. [Online]. Available: <http://blog.stevenlevithan.com/archives/regex-legends>. [Accessed: 26-Nov-2019].
- [76] N. Bazhenov, “Combining probabilistic tagging with rule-based multilevel chunking for requirements elicitation,” *Штучний інтелект*, no. 2, pp. 6–14, 2010.
- [77] C. Huertas and R. Juárez-Ramírez, “NLARE, a natural language processing tool for automatic requirements evaluation,” *Proceedings of the CUBE International Information Technology Conference*, pp. 371–378, 2012.

- [78] C. Jacob and R. Harrison, “Retrieving and analyzing mobile apps feature requests from online reviews,” *Proceedings of the 10th Working Conference on Mining Software Repositories*, pp. 41–44, 2013.
- [79] P. R. Américo Sampaio , Neil Loughran , Awais Rashid, “Mining Aspects in Requirements,” *Early Aspects 2005: Aspect-Oriented Requirements Engineering and Architecture Design Workshop*, 2005.
- [80] M. G. Ilieva and O. Ormandjieva, “Automatic transition of natural language software requirements specification into formal presentation,” *Proceedings of the 10th International Conference on Natural Language Processing and Information Systems*, pp. 392–397, 2005.
- [81] A. Nikora, J. Hayes, and E. Holbrook, “Experiments in Automated Identification of Ambiguous Natural-Language Requirements,” *21st IEEE International Symposium on Software Reliability Engineering*, 2010.
- [82] A. Sinha, S. M. Sutton, and A. Paradkar, “Text2Test: Automated inspection of natural language use cases,” *Proceedings of the 2010 Third International Conference on Software Testing, Verification and Validation*, pp. 155–164, 2010.
- [83] H. Yang, A. Willis, A. De Roeck, and B. Nuseibeh, “Automatic detection of nocuous coordination ambiguities in natural language requirements,” *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, pp. 53–62, 2010.
- [84] A. Umber, I. S. Bajwa, and M. Asif Naeem, “NL-based automated software requirements elicitation and specification,” *International Conference on Advances in Computing, Communications and Informatics*, pp. 30–39, 2011.
- [85] H. Meth, A. Maedche, and M. Einoeder, “Is knowledge power? The role of knowledge in automated requirements elicitation,” *Proceedings of the 25th International Conference on Advanced Information Systems Engineering*, pp. 578–593, 2013.
- [86] M. Elallaoui, K. Nafil, and R. Touahni, “Automatic generation of UML sequence diagrams from user stories in Scrum process,” *2015 10th International Conference on Intelligent Systems: Theories and Applications (SITA)*, pp. 1–6, 2015.

- [87] S. Gulia and T. Choudhury, “An efficient automated design to generate UML diagram from Natural Language Specifications,” *2016 6th International Conference - Cloud System and Big Data Engineering (Confluence)*, pp. 641–648, 2016.
- [88] S. McIlroy, N. Ali, H. Khalid, and A. E. Hassan, “Analyzing and automatically labelling the types of user issues that are raised in mobile app reviews,” *Empirical Software Engineering*, vol. 21, no. 3, pp. 1067–1106, 2016.
- [89] S. Keertipati, B. T. R. Savarimuthu, and S. A. Licorish, “Approaches for prioritizing feature improvements extracted from app reviews,” *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*, 2016.
- [90] Y. Tian, M. Nagappan, D. Lo, and A. E. Hassan, “What are the characteristics of high-rated apps? A case study on free Android Applications,” *Proceedings of the 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 301–310, 2015.
- [91] H. Khalid, E. Shihab, M. Nagappan, and A. E. Hassan, “What do mobile app users complain about?,” *IEEE Software*, vol. 32, no. 3, pp. 70–77, 2015.
- [92] T. P. Liang, X. Li, C. T. Yang, and M. Wang, “What in Consumer Reviews Affects the Sales of Mobile Apps: A Multifacet Sentiment Analysis Approach,” *International Journal of Electronic Commerce*, vol. 20, no. 2, pp. 236–260, 2015.
- [93] X. Gu and S. Kim, “What parts of your apps are loved by users?,” *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 760–770, 2015.
- [94] N. Chen, J. Lin, S. C. H. Hoi, X. Xiao, and B. Zhang, “AR-miner: mining informative reviews for developers from mobile app marketplace,” *Proceedings of the 36th International Conference on Software Engineering*, pp. 767–778, 2014.
- [95] W. Maalej and H. Nabil, “Bug report, feature request, or simply praise? On automatically classifying app reviews,” *2015 IEEE 23rd International Requirements Engineering Conference (RE)*, pp. 116–125, 2015.

- [96] J. Oh, D. Kim, U. Lee, J.-G. Lee, and J. Song, “Facilitating developer-user interactions with mobile app review digests,” *CHI '13 Extended Abstracts on Human Factors in Computing Systems*, pp. 1809–1814, 2013.
- [97] S. Panichella, A. Di Sorbo, and E. Guzman, “How Can I Improve My App? Classifying User Reviews for Software Maintenance and Evolution,” *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 281–290, 2015.
- [98] E. Guzman and W. Maalej, “How Do Users Like This Feature? A Fine Grained Sentiment Analysis of App Reviews,” *2014 IEEE 22nd International Requirements Engineering Conference (RE)*, pp. 153–162, 2014.
- [99] H. Yang and P. Liang, “Identification and Classification of Requirements from App User Reviews,” *Proceedings of the 27th International Conference on Software Engineering and Knowledge Engineering*, 2015.
- [100] H. Khalid, “On identifying user complaints of iOS apps,” *Proceedings of the 2013 International Conference on Software Engineering*, pp. 1474–1476, 2013.
- [101] P. M. Vu, H. V. Pham, T. T. Nguyen, and T. T. Nguyen, “Phrase-based extraction of user opinions in mobile app reviews,” *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, pp. 726–731, 2016.
- [102] M. Horridge *et al.*, “A Practical Guide To Building OWL Ontologies Using Protégé 4 and CO-ODE Tools Edition 1.3,” 2011. [Online]. Available: http://mowl-power.cs.man.ac.uk/protegeowltutorial/resources/ProtegeOWLTutorialP4_v1_3.pdf. [Accessed: 05-Jan-2020].
- [103] J. Davies, R. Studer, and P. Warren, *Semantic Web technologies: trends and research in ontology-based systems*. John Wiley & Sons., 2006.
- [104] Y. K. Hooi, M. F. Hassan, and A. M. Shariff, “A survey on ontology mapping techniques,” *Advances in Computer Science and its Applications: CSA 2013*, vol. 279 LNEE, pp. 829–836, 2014.
- [105] O. Gilson, N. Silva, P. W. Grant, and M. Chen, “From Web data to visualization via

- ontology mapping,” *Computer Graphics Forum*, vol. 27, no. 3, pp. 959–966, 2008.
- [106] F. Lin and K. Sandkuhl, “A survey of exploiting WordNet in ontology matching,” *IFIP International Conference on Artificial Intelligence in Theory and Practice*, pp. 341–350, 2008.
- [107] G. Hirst and A. Budanitsky, “Evaluating WordNet-based measures of lexical semantic relatedness,” *Computational Linguistics*, vol. 32, no. 1, pp. 13–47, 2006.
- [108] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky, “The Stanford CoreNLP Natural Language Processing Toolkit,” *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pp. 55–60, 2014.
- [109] Universal Dependencies contributors., “Universal Dependencies,” 2017. [Online]. Available: <https://universaldependencies.org/u/dep/all.html>. [Accessed: 19-Sep-2019].
- [110] R. S. Pressman and B. R. Maxim, *Software Engineering: A Practitioner’s Approach.*, 8th ed. McGraw Hill, 2015.
- [111] I. Sommerville, *Software Engineering*, 7th ed. Addison Wesley, 2004.
- [112] G. Kotonya and I. Sommerville, *Requirements engineering: processes and techniques.* Wiley Publishing., 1998.
- [113] Statistics How To, “Sample Size in Statistics (How to Find it): Excel, Cochran’s Formula, General Tips,” 2019. [Online]. Available: <https://www.statisticshowto.datasciencecentral.com/probability-and-statistics/find-sample-size/>. [Accessed: 01-Nov-2019].
- [114] FluidSurveys Team, “Calculating the Right Survey Sample Size,” 2014. [Online]. Available: <http://fluidsurveys.com/university/calculating-right-survey-sample-size/>. [Accessed: 31-Oct-2019].
- [115] Python Software Foundation, “random — Generate pseudo-random numbers,” 2019, 2019. [Online]. Available: <https://docs.python.org/3/library/random.html>. [Accessed: 04-Nov-2019].

- [116] J. Zobel, *Writing for computer science 3rd edition*. Springer, 2014.
- [117] W. Martin, M. Harman, Y. Jia, F. Sarro, and Y. Zhang, “The app sampling problem for app store mining,” *IEEE International Working Conference on Mining Software Repositories*, pp. 123–133, 2015.
- [118] M. Amith *et al.*, “Architecture and usability of OntoKeeper, an ontology evaluation tool,” *BMC Medical Informatics and Decision Making*, vol. 19:152, 2019.
- [119] Ontology Engineering Group, “OntOlogy Pitfall Scanner!,” 2019. [Online]. Available: <http://oops.linkeddata.es/advanced.jsp>. [Accessed: 08-Nov-2019].
- [120] I. Jimborean and A. Groza, “Ranking ontologies in the Ontology Building Competition BOC 2014,” *2014 IEEE 10th International Conference on Intelligent Computer Communication and Processing (ICCP)*, pp. 75–82, 2014.
- [121] S. Tartir and I. B. Arpinar, “Ontology evaluation and ranking using OntoQA,” *International Conference on Semantic Computing (ICSC 2007)*, pp. 185–192, 2007.
- [122] J. Xu and X. Ma, “A web-based ontology evaluation system,” *2008 International Conference on Advanced Language Processing and Web Information Technology*, pp. 104–107, 2008.
- [123] H. Zhu, D. Liu, I. Bayley, A. Aldea, Y. Yang, and Y. Chen, “Quality model and metrics of ontology for semantic descriptions of web services,” *Tsinghua Science and Technology*, vol. 22, no. 3, pp. 254–272, 2017.
- [124] A. Groza, I. Dragoste, I. Sincai, I. Jimborean, and V. Moraru, “An ontology selection and ranking system based on the analytic hierarchy process,” *2014 16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pp. 293–300, 2014.
- [125] D. Wisniewski, J. Potoniec, A. Lawrynowicz, and C. M. Keet, “Competency Questions and SPARQL-OWL Queries Dataset and Analysis,” arXiv:1811.09529, 2018.
- [126] S. Kralin, “Question on Snap SPARQL plugin on Stackoverflow,” 2017. [Online]. Available: <https://stackoverflow.com/questions/44805204/sparql-query-individual->

in-protege. [Accessed: 09-Nov-2019].

- [127] ontotext, “What are Ontologies?,” 2019. [Online]. Available: <https://www.ontotext.com/knowledgehub/fundamentals/what-are-ontologies/>. [Accessed: 12-Nov-2019].
- [128] Jscher2000, “How do I install an earlier version of FF?,” 2019. [Online]. Available: <https://support.mozilla.org/en-US/questions/1109945>. [Accessed: 19-Aug-2019].
- [129] Mozilla, “Index of /pub/firefox/releases/,” 2019. [Online]. Available: <https://ftp.mozilla.org/pub/firefox/releases/>. [Accessed: 19-Aug-2019].
- [130] AutomatedTester, “geckodriver,” 2016. [Online]. Available: <https://github.com/mozilla/geckodriver/releases/tag/v0.11.1>. [Accessed: 15-Dec-2019].
- [131] Oracle Corporation and/or its affiliates, “Chapter 2 Installing and Upgrading MySQL,” 2019. [Online]. Available: <https://dev.mysql.com/doc/refman/5.7/en/installing.html>. [Accessed: 15-Dec-2019].
- [132] Oracle Corporation and/or its affiliates, “Chapter 2 Installation,” 2019. [Online]. Available: <https://dev.mysql.com/doc/workbench/en/wb-installing.html>. [Accessed: 15-Dec-2019].
- [133] Microsoft, “Downloads,” 2019. [Online]. Available: <https://visualstudio.microsoft.com/downloads/>. [Accessed: 15-Dec-2019].
- [134] Stanford NLP Group, “Download,” 2018. [Online]. Available: <https://nlp.stanford.edu/software/lex-parser.shtml#Download>. [Accessed: 15-Dec-2019].
- [135] Stanford NLP Group, “What does an UnsupportedClassVersionError mean?,” 2019. [Online]. Available: <https://nlp.stanford.edu/software/parser-faq.html#l>. [Accessed: 15-Dec-2019].

8 Appendix A Software installation

This framework needs some software to run. The main software is FireFox version 47, Python 3, selenium, NLTK, MySQL, SO-CAL, and Stanford Parser. Specifically, the web sourcing code needs the FireFox browser version 47 and Geckodriver v0.11.1.

The specific versions for the major software in this implementation code are below:

Table 47 Software versions that work in this implementation

Software	Version
FireFox browser	47
Python	Latest version 3.x (32-bit) (3.7 – 32 bit)
NLTK	Latest version with All packages (3.4.5)
Selenium	Latest version (3.141.0)
Geckodriver	v0.11.1
Stanford Parser	Latest version (3.9.2)
JDK	Latest version (13)
MySQL Server and Workbench	Latest version (8.0)

In the “Version” column above, the versions in the brackets are the current versions working on the computer where this prototype has been developed and evaluated.

Since the framework is implemented on a Windows 10 computer, and verified on a Windows 7 computer, the installation suggestions here will take Windows system as the main example. Future researchers are encouraged to adapt the prototype to other operating systems. There are no major obstacles in this prototype’s code preventing this.

8.1 Firefox 47 installation [128]

- A) Download Firefox 47 from the official website [129].
- B) Exit Firefox if it is open.
- C) Rename the existing Firefox program folder (if exists) as follows:

(32-bit Firefox on 64-bit Windows)

C:\Program Files (x86)\Mozilla Firefox

to

C:\Program Files (x86)\Fx68

(Other versions)

C:\Program Files\Mozilla Firefox

to

C:\Program Files\Fx68

- D) Run the installer you downloaded in step A). It should automatically connect to your existing settings.

Note: When being prompted for update to a newer version after the installation, please do not accept.

- E) Open Firefox and click “Options” at the bottom of the screen. In “Advanced - Update”, select “Never check for updates” for “Firefox updates”. This will prevent Firefox from downloading updates in the future.

Some plugins may exist only in that Fx56 folder. If something essential is missing, look in these folders:

- \Fx56\Plugins
- \Fx56\browser\extensions

However, if the purpose of your Firefox is merely for this web sourcing code to work, you do not need to do anything for the plugins.

8.2 The geckodriver installation

A geckodriver.exe is needed for Firefox browser to use Selenium3.

Specifically, geckodriver v0.11.1 is used in this implementation, which bypasses the problem of connection being refused sometimes and provides very quick speed for the web sourcing code in execution.

For geckodriver download, please go to the geckodriver page of version v0.11.1 [130].

Selenium client bindings will try to locate the geckodriver executable from the system PATH. Thereby the full directory path to it has to be added in the Path system variable (environment variable in Windows). Restarting the computer is needed after the PATH updating.

8.3 MySQL, and supporting Visual Studio, Python 3 (64-bit) installation

There is more than one way to install MySQL on Windows, such as through WAMP installation. WampServer contains Apache, MySQL, and PHP for Windows. This introduction here is for MySQL Server and MySQL Workbench installation through MySQL official website.

For MySQL Server installation, please refer to the Installing and Upgrading MySQL page [131] and choose the appropriate operation system.

For MySQL Workbench installation, please refer to the Installing page of MySQL Workbench [132] and choose the appropriate operation system.

On a Windows computer, there is an option to install both MySQL Server and MySQL Workbench together using the MySQL Installer, which is introduced above.

If this option is taken, the setup type for the MySQL installer community MSI Installer can be set as the default setting, Developer Default.

During this installation, Visual Studio and Python 3 are required to be installed.

For Visual Studio download, please refer to the download page of Visual Studio [133] and choose the appropriate version.

At the time of September 2019, the latest MySQL Installer 8.0.17 does not accept the latest version of Visual Studio 2019 on a Windows 7 computer. Thereby, an older version of Visual Studio 2012, 2013, 2015, or 2017 needs to be installed through the “Older versions” link at the lower part of the above page.

Installing a previous version of Visual Studio, a Microsoft account might be needed. A subscription to “Visual Studio Dev Essentials” might also be needed, which is permanently free.

During the installation of Visual Studio, Python 3 can be selected to install together. However, the version of the included Python 3 may not suit the requirement of MySQL. If this happens, for the proper version of Python installation, please follow the link suggested by the MySQL installer.

After the successful installation, depending on the MySQL version, the “MySQL Command Line Client” could be under the “MySQL” menu item under the “Start” → “All Programs”, or further down under the “MySQL Server 8.0”. “MySQL Command Line Client” is the convenient way to use for the purpose of database manual operation and testing.

MySQL Workbench is useful to create database users, grant privileges, import and export database backups.

8.4 Python 3 (32-bit) and libraries such as NLTK installation

For NLTK installation, please refer to the official NLTK documentation [66] and choose the appropriate operation system environment.

When installing Python 3 (32-bit) from this page, it would be good to choose “Customize installation” and make sure to install pip.

Depending on the version of installed pip, pip might need to be upgraded. If it is needed, it is very important to run the command as “Administrator” in Windows CMD.

Installing NLTK Data is necessary. It is recommended to install all packages of NLTK.

Run Windows CMD as administrator:

```
cd C:\Program Files (x86)\Python36-32\Scripts (the Python Pip installation path)
```

```
pip install nltk
```

In Python 3:

```
import nltk
```

```
nltk.download()
```

This command will bring up the interactive window. From the list, please choose to download “all” for all packages, and then click the “download” button. This will download all tokenizers, chunkers, other algorithms and all of the corpora.

After this, libraries can be installed with “pip install” now in CMD, such as:

```
pip install pymysql
```

8.5 Selenium installation

To install Selenium in Python:

Run Windows CMD as Administrator:

```
cd C:\Program Files (x86)\Python36-32\Scripts
```

```
pip install selenium
```

To install any versions of Selenium in Python:

```
pip uninstall selenium
```

```
pip install selenium==3.3.1
```

To retrieve the version of Selenium currently installed, from Python:

```
>>> import selenium
```

```
>>> selenium.__version__
```

```
'3.141.0'
```

8.6 graphviz installation

Run Windows CMD as Administrator:

```
cd C:\Program Files (x86)\Python36-32\Scripts
```

```
pip install graphviz
```

Successfully installed graphviz-0.13

Graphviz relies on Tree to produce a text file of the trees, which needs to be installed if the text file of the trees is of interest for future researchers.

```
pip install ete3
```

Successfully installed ete3-3.1.1

8.7 Stanford Parser installation

For Stanford Parser download, please go to the Stanford Parser official website [134]. (For previous versions, such as Version 3.8.0, please refer to the Release history on the same page.)

Extract the content of the zip file into a permanent folder that is easy to manage, for example C:\Python37-32.

When using the Stanford Parser that you have downloaded in a piece of Python code, such as the list of code C.13, please use the following lines of code to point to it:

```
os.environ["CLASSPATH"] = 'C:\Program Files (x86)\Python37-32\stanford-parser-full-2018-10-17\stanford-parser.jar'
```

```
os.environ["STANFORD_MODELS"] = 'C:\Program Files (x86)\Python37-32\stanford-parser-full-2018-10-17\stanford-parser-3.9.2-models.jar'
```

```
os.environ["JAVAHOME"] = 'C:\Program Files\Java\jdk-13'
```


It is worth noting, Stanford Parser relies on Java JDK. If the JDK version is not recent enough, Stanford Parser refuses to run and could give a message for unsupported class version error. But this is not the case. The real reason is that the JDK is out of date. Installing the latest version of JDK and pointing to it in the above line of code will fix the problem.

Please see the explanation that The Stanford Natural Language Processing Group gives in their FAQ [135].

18. What does an `UnsupportedClassVersionError` mean?

If you see the error:

```
Exception in thread "main" java.lang.UnsupportedClassVersionError:  
edu.stanford.nlp.parser.lexparser.LexicalizedParser (Unsupported major.minor version xy.z)
```

it means that you don't have a recent enough version of Java installed. If "xy.z" is "49.0", then you don't have Java 5 installed. If "xy.z" is "52.0", then you don't have Java 8 installed. Etc. You should upgrade at <http://www.oracle.com/technetwork/java/javase/downloads/>.

Figure 33 Stanford NLP Group explains the JDK version issue

9 Appendix B Steps to run this framework

Table 48 Steps to run this system

1	Database creation.	C.1
2	Data sourcing from Google Play.	C.2, C.3, C.4
2.a.	Alternatively, restoring a database from a backup file.	MySQL Workbench
3	Running SO-CAL.	C.5, Adapted SO-CAL
4	Eliciting user requests.	C.8
5	Extracting topic-opinion pairs.	C.9 or C.10
6	Filtering the topic-opinion pairs with keywords mapped to ontology.	C.11 or C.12
7	Ontology population.	C.15
8	Running SPARQL Queries for various questions.	
9	Further data visualization (optional for the users)	

10 Appendix C POS tag list and user request rule convention

POS tag list:

CC	coordinating conjunction
CD	cardinal digit
DT	determiner
EX	existential there (like: "there is" ... think of it like "there exists")
FW	foreign word
IN	preposition/subordinating conjunction
JJ	adjective 'big'
JJR	adjective, comparative 'bigger'
JJS	adjective, superlative 'biggest'
LS	list marker 1)
MD	modal could, will
NN	noun, singular 'desk'
NNS	noun plural 'desks'
NNP	proper noun, singular 'Harrison'
NNPS	proper noun, plural 'Americans'
PDT	predeterminer 'all the kids'
POS	possessive ending parent's
PRP	personal pronoun I, he, she
PRP\$	possessive pronoun my, his, hers
RB	adverb very, silently,
RBR	adverb, comparative better

RBS adverb, superlative best
 RP particle give up
 TO to go 'to' the store.
 UH interjection errrrrrrm
 VB verb, base form take
 VBD verb, past tense took
 VBG verb, gerund/present participle taking
 VBN verb, past participle taken
 VBP verb, sing. present, non-3d take
 VBZ verb, 3rd person sing. present takes
 WDT wh-determiner which
 WP wh-pronoun who, what
 WP\$ possessive wh-pronoun whose
 WRB wh-abverb where, when

Identifiers:

\d = any number

\D = anything but a number

\s = space

\S = anything but a space

\w = any letter

\W = anything but a letter

. = any character, except for a new line

`\b` = space around whole words

`\.` = period. must use backslash, because `.` normally means any character.

Modifiers:

`{1,3}` = for digits, u expect 1-3 counts of digits, or "places"

`+` = match 1 or more

`?` = match 0 or 1 repetitions.

`*` = match 0 or MORE repetitions

`$` = matches at the end of string

`^` = matches start of a string

`|` = matches either/or. Example `x|y` = will match either x or y

`[]` = range, or "variance"

`{x}` = expect to see this amount of the preceding code.

`{x,y}` = expect to see this x-y amounts of the precedng code

White Space Charts:

`\n` = new line

`\s` = space

`\t` = tab

`\e` = escape

`\f` = form feed

\r = carriage return

Characters to REMEMBER TO ESCAPE IF USED!

. + * ? [] \$ ^ () { } | \

Brackets:

[] = quant[ia]tative = will find either quantitative, or quantatative.

[a-z] = return any lowercase letter a-z

[1-5a-qA-Z] = return all numbers 1-5, lowercase letters a-q and uppercase A-Z

11 Appendix D Topic-Opinion pair extraction linguistic rules

[65, 'acl', 'JJ', 'NN', 'parent'],

[64, 'acl', 'JJ', 'VBG', 'parent'],

[45, 'acl', 'VBN', 'JJS', 'child'],

[102, 'acl', 'VBG', 'NN', 'child'],

[162, 'acl', 'VBG', 'NNP', 'child'],

[73, 'acl:relcl', 'JJ', 'PRP', 'parent'],

[149, 'acl:relcl', 'VBG', 'NN', 'parent'],

[125, 'acl:relcl', 'VBN', 'NNP', 'child'],

[21, 'acl:relcl', 'VBZ', 'NN', 'parent'],

[122, 'advcl', 'VB', 'JJ', 'child'],

[123, 'advcl', 'VBG', 'JJ', 'child'],

[85, 'advcl', 'VBG', 'VBD', 'child'],

[147, 'advmod', 'RB', 'CD', 'parent'],

[11, 'advmod', 'RB', 'JJ', 'child'],

[128, 'advmod', 'RB', 'VB', 'parent'],

[94, 'advmod', 'RB', 'VBD', 'parent'],

[68, 'advmod', 'RB', 'VBG', 'parent'],

[40, 'advmod', 'RB', 'VBN', 'parent'],

[25, 'advmod', 'RB', 'VBP', 'parent'],

[148, 'advmod', 'RBR', 'NNS', 'parent'],

[151, 'advmod', 'RBR', 'VBZ', 'parent'],

[160, 'advmod', 'RBS', 'VBN', 'parent'],

[78, 'amod', 'JJ', 'JJ', 'parent'],

[49, 'amod', 'JJ', 'NN', 'parent'],

[31, 'amod', 'JJ', 'NNP', 'parent'],

[120, 'amod', 'JJ', 'NNPS', 'parent'],

[105, 'amod', 'JJ', 'NNS', 'parent'],

[92, 'amod', 'JJ', 'PRP', 'parent'],

[26, 'amod', 'JJR', 'NN', 'parent'],

[88, 'amod', 'JJR', 'NNS', 'parent'],

[113, 'amod', 'JJS', 'NN', 'parent'],

[150, 'case', 'IN', 'PRP', 'parent'],

[14, 'case', 'IN', 'VBG', 'parent'],

[117, 'ccomp', 'VB', 'VB', 'child'],

[22, 'ccomp', 'VB', 'VBG', 'child'],

[104, 'ccomp', 'VB', 'VBN', 'child'],

[97, 'ccomp', 'VB', 'VBP', 'child'],

[57, 'ccomp', 'VB', 'VBZ', 'child'],

[18, 'ccomp', 'VBZ', 'VBP', 'child'],

[106, 'compound', 'NNP', 'NNP', 'parent'],

[124, 'compound', 'NNP', 'NNPS', 'parent'],

[76, 'dep', 'JJ', 'NN', 'child'],

[56, 'dep', 'JJ', 'VB', 'parent'],

[27, 'dep', 'JJ', 'VBP', 'parent'],

[87, 'dep', 'NN', 'JJ', 'child'],

[52, 'dep', 'NN', 'JJS', 'child'],

[115, 'dep', 'NN', 'NN', 'child'],

[59, 'dep', 'NN', 'NNP', 'child'],

[109, 'dep', 'NN', 'VBP', 'child'],

[84, 'dep', 'NNS', 'JJS', 'child'],

[35, 'dep', 'VB', 'JJ', 'child'],

[36, 'dep', 'VB', 'JJS', 'child'],

[69, 'dep', 'VBG', 'JJ', 'child'],

[164, 'dep', 'VBG', 'JJR', 'child'],

[75, 'dep', 'VBG', 'NN', 'child'],

[126, 'dep', 'VBG', 'NNP', 'child'],

[139, 'dep', 'VBP', 'VBG', 'child'],

[114, 'dobj', 'NN', 'VB', 'child'],

[131, 'dobj', 'NN', 'VBG', 'child'],

[38, 'dobj', 'NN', 'VBP', 'child'],

[141, 'dobj', 'NN', 'VBZ', 'child'],

[20, 'dobj', 'NNPS', 'VBZ', 'child'],

[98, 'dobj', 'NNS', 'VB', 'child'],
[130, 'dobj', 'NNS', 'VBP', 'child'],
[159, 'dobj', 'PRP', 'VB', 'child'],
[143, 'dobj', 'PRP', 'VBD', 'child'],
[30, 'dobj', 'PRP', 'VBP', 'child'],
[129, 'dobj', 'PRP', 'VBZ', 'child'],
[61, 'dobj', 'VBG', 'VBN', 'child'],

[110, 'nmod', 'NN', 'JJ', 'child'],
[134, 'nmod', 'NN', 'NNP', 'child'],
[132, 'nmod', 'NNS', 'JJ', 'child'],
[72, 'nmod', 'NNS', 'NNS', 'child'],
[77, 'nmod', 'NNS', 'VBN', 'child'],
[142, 'nmod', 'PRP', 'JJ', 'child'],
[163, 'nmod', 'PRP', 'NN', 'child'],

[54, 'nmod:poss', 'PRP', 'NN', 'child'],

[158, 'nsubj', 'JJ', 'VBP', 'parent'],
[127, 'nsubj', 'JJS', 'VBP', 'parent'],
[23, 'nsubj', 'JJS', 'VBZ', 'parent'],
[103, 'nsubj', 'NN', 'JJ', 'child'],
[112, 'nsubj', 'NN', 'JJR', 'child'],
[70, 'nsubj', 'NN', 'RBR', 'child'],

[144, 'nsubj', 'NN', 'VBG', 'child'],
[29, 'nsubj', 'NN', 'VBN', 'child'],
[37, 'nsubj', 'NN', 'VBZ', 'child'],
[133, 'nsubj', 'NNP', 'JJ', 'child'],
[46, 'nsubj', 'NNP', 'NNP', 'child'],
[145, 'nsubj', 'NNP', 'VBD', 'child'],
[135, 'nsubj', 'NNP', 'VBD', 'parent'],
[152, 'nsubj', 'NNP', 'VBG', 'child'],
[108, 'nsubj', 'NNP', 'VBP', 'parent'],
[81, 'nsubj', 'NNP', 'VBZ', 'child'],
[153, 'nsubj', 'NNP', 'VBZ', 'parent'],
[24, 'nsubj', 'NNS', 'JJ', 'child'],
[107, 'nsubj', 'NNS', 'JJS', 'child'],
[161, 'nsubj', 'NNS', 'VBP', 'child'],
[74, 'nsubj', 'NNS', 'VBZ', 'child'],
[111, 'nsubj', 'PRP', 'JJ', 'child'],
[156, 'nsubj', 'PRP', 'NN', 'child'],
[32, 'nsubj', 'PRP', 'VBN', 'child'],
[60, 'nsubj', 'PRP', 'VBP', 'child'],
[34, 'nsubj', 'PRP', 'VBZ', 'child'],

[41, 'nsubjpass', 'NN', 'NN', 'child'],
[80, 'nsubjpass', 'NN', 'VBN', 'child'],
[95, 'nsubjpass', 'PRP', 'JJR', 'child'],

[58, 'nsubjpass', 'PRP', 'NN', 'child'],

[119, 'xcomp', 'JJ', 'VB', 'parent'],

[121, 'xcomp', 'JJ', 'VBZ', 'parent'],

[100, 'xcomp', 'JJR', 'VBG', 'parent'],

[28, 'xcomp', 'JJR', 'VBP', 'parent'],

[154, 'xcomp', 'NN', 'VBG', 'parent'],

[165, 'xcomp', 'RBR', 'VBN', 'parent'],

[15, 'xcomp', 'VB', 'JJ', 'child'],

[137, 'xcomp', 'VB', 'JJR', 'child'],

[71, 'xcomp', 'VB', 'VB', 'child'],

[62, 'xcomp', 'VB', 'VBP', 'child'],

[48, 'xcomp', 'VB', 'VBZ', 'child'],

[44, 'xcomp', 'VBG', 'JJ', 'child'],

[17, 'xcomp', 'VBG', 'VBP', 'child'],

[47, 'xcomp', 'VBN', 'VB', 'child'],

12 Appendix E Popular Opinions extracted from dataset one and two.

'1st', '1-star', '2-star', '3-star', '4-star', '5-star',

'able', 'abandon', 'abruptly', 'absent', 'absurd', 'accept', 'acceptable', 'acesible', 'accessible',
'accountable', 'accumulative', 'accurate', 'accurately', 'accustomed', 'achievable', 'achieved',
'achieving',

'achievements', 'active', 'actively', 'acurately', 'acutely', 'adaptable', 'addicted', 'addictive',
'adequate', 'adequately', 'ad-free', 'ads-free', 'advantage', 'advantageous', 'advantages',
'adviaeeable',

'adjustable', 'adorable', 'adores', 'advanced', 'affected', 'affecting', 'affection', 'affective',
'affordable', 'aggravates', 'aggressive', 'agree', 'aid', 'aimlessly', 'allow', 'allows', 'alright',

'alternative', 'amaaaazing', 'amazed', 'amazing', 'ambiguous', 'ambitious', 'analytic',
'analytical', 'angry', 'annoy', 'annoyed', 'annoying', 'annoyingly', 'annyoing', 'anoying',
'antisocially',

'appealing', 'appreciate', 'appreciated', 'appreciates', 'appreciative', 'arrogant', 'ashamed',
'assistant', 'assists', 'astonishing', 'astounding', 'attractive', 'augmented', 'authentic', 'automatic',

'automatically', 'auto-pause', 'auto-paused', 'auto-pauses', 'autofill', 'autofills', 'automated',
'autopause', 'autopaused', 'autopauses', 'average', 'averages', 'avoid', 'awesome', 'awesomely',

'awesomeness', 'awful', 'awkward', 'awrsome', 'awsome',

'bad', 'balanced', 'bare', 'barely', 'bearable', 'beastly', 'beat', 'beating', 'beats', 'beautiful',
'beautifull', 'beautifully', 'beloved', 'benefical', 'beneficial', 'benefit', 'benefited', 'benefiting',

'benifical', 'best', 'better', 'big', 'bigger', 'biggest', 'bizarrely', 'blame', 'blames', 'blessing',
'blind', 'bloated', 'block', 'blocked', 'blocking', 'blocks', 'borderline', 'bore', 'bored', 'boring',

'bother', 'bottom', 'brainless', 'breaker', 'breaking', 'briefly', 'bright', 'brill', 'brilliant',
'brilliantly', 'brisk', 'brittle', 'broad', 'broken', 'brutal', 'buggy', 'bugs', 'bulky', 'busy', 'buy',

'capable', 'capably', 'captivate', 'captivating', 'care', 'careful', 'carefully', 'cares', 'caring', 'central', 'challenged', 'challenging', 'changer', 'chaotic', 'chargeable', 'charitable', 'cheap', 'cheaper',

'cheapest', 'cheated', 'childish', 'chronological', 'chronologically', 'chunky', 'classy', 'clean', 'clear', 'clearer', 'clearly', 'clever', 'clinical', 'clumsy', 'clunky', 'colorful', 'comfortable',

'commendable', 'commercial', 'common', 'comparable', 'compartable', 'compatible', 'compelling', 'competative', 'competent', 'competetive', 'competing', 'competition', 'competitions', 'competitive',

'competitively', 'competitor', 'compicated', 'complain', 'complaining', 'complaint', 'complement', 'complements', 'complete', 'completed', 'complex', 'complexe', 'complicate', 'complicated', 'comprehensive',

'compromised', 'compulsive', 'compulsory', 'concentrate', 'concentrated', 'concentrates', 'concerned', 'concise', 'concisely', 'concurrently', 'condensed', 'confident', 'configurable', 'conflicted',

'conflicting', 'confuse', 'confused', 'confuses', 'confusing', 'confusion', 'congrats', 'congratulate', 'congratulations', 'conscious', 'conservative', 'considerable', 'considerably', 'considering',

'consistent', 'consistently', 'consolidate', 'constant', 'constructive', 'consuming', 'continuous', 'continously', 'continually', 'continuously', 'contrast', 'controlled', 'controlling', 'convenience',

'convenient', 'conveniently', 'convient', 'convinced', 'convincing', 'convinient', 'cool', 'correct', 'corrected', 'correction', 'corrective', 'correctly', 'correlate', 'correlates', 'corrupt', 'cos',

'cosas', 'counter-intuitive', 'countless', 'covenant', 'cracks', 'cramped', 'crap', 'crappy', 'crash', 'crashed', 'crashes', 'crashing', 'crazy', 'creative', 'creepy', 'critical', 'crowded', 'crucial',

'crummy', 'crush', 'crushed', 'crushes', 'crushing', 'cumbersome', 'cumulative', 'current', 'customisable', 'customised', 'customizable', 'cute', 'cutesy',

'damn', 'dangerous', 'dated', 'daunting', 'dead', 'deadlinks', 'debilitating', 'decent', 'dedicated', 'defenetly', 'defiantly', 'definitive', 'delete', 'deleting', 'delighted', 'demonstrates', 'demonstrations',

'demoralising', 'demonstration', 'dependable', 'dependent', 'deserve', 'deserved', 'deserves', 'deserving', 'desirable', 'desire', 'desired', 'desperate', 'desperately', 'destroyed', 'destroys', 'detailed',

'different', 'difficult', 'diligently', 'difficulty', 'diligent', 'direct', 'directly', 'disappoint', 'disappointed', 'disappointing', 'disappointments', 'disappoints', 'disaster', 'disciplined', 'discourage',

'discouraged', 'discouraging', 'discrepancies', 'disentangle', 'disgusted', 'disheartened', 'disheartening', 'dishonorably', 'disillusioned', 'dislike', 'disorganized', 'displaced', 'displeased', 'dissatisfied',

'dissatisfying', 'dissatisfied', 'distorted', 'distracting', 'do-able', 'doable', 'dodgy', 'dominate', 'doubt', 'dreaded', 'dreadful', 'dream', 'dry', 'dubious', 'dull', 'dumb', 'duplicate', 'duplicated',

'easy', 'easily', 'eagerly', 'earlier', 'early', 'ease', 'eases', 'easier', 'easiest', 'easily', 'easy', 'easier', 'easy-to-use', 'easy', 'ecstatic', 'editable', 'educational', 'educative', 'effective',

'effectively', 'effectively', 'efficient', 'efficiently', 'effortless', 'effortlessly', 'elegant', 'elegantly', 'elevated', 'elite', 'elude', 'embarrassed', 'embarrassing', 'emotional', 'empowering', 'empty',

'encourage', 'encouraged', 'encourages', 'encouraging', 'endearing', 'endless', 'endlessly', 'enhanced', 'enjoyed', 'enlarged', 'enlighten', 'enjoy', 'enjoyable', 'enjoying', 'enormously', 'enough', 'entertain',

'entertained', 'entertaining', 'enthralled', 'enthraling', 'enthusiastic', 'equal', 'erratic', 'erratically', 'essential', 'establish', 'evenly', 'exaggerated', 'excel', 'excellent', 'excels', 'exceptional',

'excessive', 'excited', 'exciting', 'exclusively', 'excellent', 'exemplary', 'exhausted', 'exhausting', 'exhilarating', 'exorbitant', 'exotic', 'expensive', 'experienced', 'explicit', 'exportable', 'extensive',

'extra', 'extraordinary', 'extreme', 'eye-catching', 'evert', 'everything', 'everywhere', 'evident', 'evil', 'exceeds', 'excellent', 'excellent', 'excludes', 'expedient',

'fab', 'fabulous', 'fabulously', 'facilitates', 'fail', 'fails', 'failure', 'faint', 'fair', 'fairly', 'fake', 'falls', 'false', 'familiar', 'famous', 'fanatic', 'fancier', 'fanciful', 'fancy', 'fantastic', 'fast',

'fat', 'fatigued', 'fault', 'faulty', 'favorable', 'favourable', 'favorite', 'favourite', 'fewer', 'fiddly', 'fine', 'finer', 'finest', 'finicky', 'first', 'fit', 'fits', 'fixed', 'flaky', 'flashy', 'flat',

'flawed', 'flawless', 'flawlessly', 'flexibility', 'flexable', 'flexible', 'flooded', 'flooding', 'fluctuating', 'focused', 'fond', 'force', 'forced', 'forceful', 'forcing', 'forces', 'forever', 'forget',

'forgets', 'forgotten', 'fragile', 'frankly', 'fraudulent', 'freak', 'freaking', 'freaks', 'freaky', 'free', 'freezes', 'frequently', 'fresh', 'freshly', 'friendly', 'frustating', 'frustrated', 'frustrates',

'frustrating', 'frustration', 'fullest', 'full-screen', 'fun', 'functioning', 'fundamental', 'funded', 'funny', 'fussing',

'gaining', 'gamifies', 'gamify', 'garbage', 'geek', 'geeky', 'general', 'generic', 'generous', 'genuine', 'glad', 'gladly', 'glitchy', 'glorious', 'godsend', 'good', 'good-looking', 'gr8', 'gr8t', 'grand', 'graphic',

'grateful', 'great', 'greater', 'greatest', 'greatful', 'greatly', 'greedy', 'groovy', 'grown', 'guilty', 'gutted',

'habbits', 'habit', 'handy', 'happier', 'happily', 'happy', 'hard', 'harder', 'hardly', 'harmful', 'hassel', 'hassle', 'hate', 'hated', 'hates', 'hazardous', 'health', 'healthier', 'healthy', 'heath', 'heavily',

'hectic', 'hefty', 'hell', 'help', 'helped', 'helpful', 'helpfull', 'helpfully', 'helping', 'helps', 'hesitant', 'high', 'higher', 'highest', 'highlighted', 'hitting', 'holistic', 'honest', 'honestly', 'hope',

'hoping', 'horrendous', 'horrible', 'hot', 'huge', 'humble', 'hurting', 'hurts', 'hustle',

'ideal', 'idiotic', 'ignore', 'illogical', 'imaginable', 'immature', 'immediately', 'immensely', 'impaired', 'imperfect', 'important', 'importantly', 'impose', 'impossible', 'impressed', 'impress', 'impressed',

'impressing', 'impressive', 'improve', 'improved', 'improving', 'inaccurate', 'inaccurately', 'inactive', 'inacurate', 'inacurrate', 'inadequate', 'inapplicable', 'inappropriate', 'incapable', 'incentive',

'incentivizes', 'inclined', 'inclusive', 'incompatible', 'incomplete', 'inconsistencies', 'inconsistency', 'inconsistent', 'inconvenient', 'incorrect', 'incorrectly', 'increase', 'increasing', 'incredible',

'increment', 'independently', 'india-centric', 'indispensable', 'inevitable', 'inexistence', 'inexpensive', 'inferior', 'infested', 'infinite', 'inflated', 'inflating', 'informational', 'informative',

'infrequent', 'infrequently', 'inhibits', 'innovative', 'insane', 'inspirational', 'inspires', 'inspiring', 'insufficient', 'insulting', 'interfere', 'integrative', 'intelligent', 'intense', 'intensive',

'interested', 'interesting', 'interacting', 'interactive', 'interferes', 'interfering', 'intermediate', 'intermittent', 'interrupt', 'interrupted', 'interrupting', 'interrupts', 'interest', 'initial',

'intimidated', 'intimidating', 'intolerable', 'interesting', 'intriguing', 'intuitive', 'intuitive', 'intrusive', 'inundated', 'invaded', 'invalid', 'invaluable', 'invented', 'inventive', 'invincible',

'invisible', 'iove', 'irrelevant', 'irresponsible', 'irritated', 'irritates', 'irritating', 'jabbing', 'jaded', 'jealous', 'jittery', 'joy', 'joyful', 'joys', 'jumbles', 'jump', 'junk', 'junior', 'justify',

'keen', 'keep', 'keeping', 'keeps', 'kidding', 'killer', 'killing', 'kind', 'kindly', 'kudos',

'lack', 'lacking', 'lacklustre', 'laden', 'lag', 'lagging', 'lapsed', 'large', 'larger', 'largest', 'lasting', 'latest', 'laugh', 'laughs', 'lazy', 'laziest', 'leading', 'learned', 'least', 'legal', 'legally',

'legendary', 'legible', 'legit', 'lessens', 'lifeline', 'life-long', 'lifelong', 'lifesaver', 'light', 'lighter', 'like', 'liked', 'like-minded', 'likes', 'liking', 'limit', 'limited', 'limits', 'little',

'loaded', 'local', 'localised', 'logic', 'logical', 'lol', 'long-term', 'loose', 'loosing', 'lose', 'loss', 'losing',

'lousy', 'love', 'loveable', 'loved', 'lovely', 'lover', 'lovers', 'loves', 'loving', 'low', 'lower', 'lowest', 'luck', 'lucky', 'luv', 'luve', 'luved',

'mad', 'major', 'manageable', 'magnifying', 'malfunctioning', 'malfunctions', 'mandatory', 'many', 'marvellous', 'massive', 'massively', 'master', 'matter', 'matters', 'mature', 'maximize', 'maximum', 'meaningful',

'meaningless', 'mess', 'messes', 'messaging', 'messy', 'military', 'min', 'mindful', 'minimal', 'minimalist', 'minimalistic', 'minimised', 'miniscule', 'minor', 'minted', 'miscounts', 'misleading', 'missed',

'misses', 'missing', 'mistake', 'moble', 'moderate', 'moderately', 'modern', 'monetary', 'monotonous', 'monotony', 'moral', 'motivate', 'motivated', 'motivates', 'motivating', 'motivation', 'motivational',

'motivator', 'motivators', 'motives', 'motivate', 'moved', 'moving', 'multi-faceted', 'mundane',

'nad', 'native', 'natural', 'neat', 'necessary', 'need', 'needs', 'negates', 'negative', 'nervous', 'new', 'newer', 'newlying', 'nice', 'nicely', 'nicer', 'nifty', 'non-chronological', 'non-existent', 'non-functional',

'non-intrusive', 'non-premium', 'nonsense', 'nonstop', 'normal', 'nothing', 'noticeable', 'noticed', 'noticing', 'nowhere', 'numeric', 'numerical', 'numerous',

'obese', 'obliged', 'obnoxious', 'obscure', 'obsessed', 'obsolete', 'obstacle', 'obtuse', 'obvious', 'obviously', 'occasional', 'odd', 'old', 'ok', 'okay', 'one-time', 'onetouch', 'opened', 'opening', 'openly',

'opens', 'opposed', 'optimised', 'optimising', 'optimistic', 'optimize', 'optimized', 'optimus', 'optional', 'ordinary', 'organised', 'organize', 'organized', 'outdated', 'outstanding', 'outperforms', 'outright',

'overestimated', 'overestimates', 'overpriced', 'overwhelmed', 'overwhelming', 'packed', 'pain', 'painful', 'pathetic', 'patient', 'paying', 'peerless', 'peeved', 'perfect', 'perfectly', 'perks', 'permanente', 'permanently', 'premier', 'perplexing', 'personalised', 'personalize', 'personalized',

'pertinent', 'phenomenal', 'picked', 'pissed', 'pity', 'plain', 'playful', 'pleasant', 'pleased', 'pleasing', 'pleasure', 'plentiful', 'pointless', 'poison', 'poor', 'pop', 'pops', 'popular', 'portable',

'positive', 'potential', 'powerful', 'practical', 'precious', 'precise', 'prefer', 'preferable', 'prefers', 'pre-loaded', 'premium', 'prepared', 'pressure', 'prettier', 'pretty', 'priceless', 'pricey', 'pricy',

'programmable', 'programmed', 'primary', 'primarily', 'prime', 'printable', 'pro', 'proactive', 'problem', 'problematic', 'problems', 'productive', 'professional', 'profound', 'prominent', 'promising',

'promptly', 'proper', 'properly', 'protective', 'proud', 'proven', 'proves', 'psychological', 'public', 'puny', 'pure', 'purposeful', 'pushy',

'questionable', 'quick', 'quicker', 'quickly', 'quiet', 'quit',

'raising', 'rapid', 'rare', 'rarely', 'raw', 'readable', 'ready', 'realistic', 'real-time', 'reasonable', 'reasonably', 'reccomend', 'reccomended', 'recomended', 'recommend', 'recommended', 'recommended',

'recommending', 'recommends', 'recreational', 'rendered', 'reduce', 'refuse', 'refuses', 'regular', 'regularly', 'relaxing', 'relevant', 'reliable', 'reliably', 'relieved', 'religiously', 'remarkable',

'remarkably', 'reputable', 'responsible', 'responsive', 'restore', 'restored', 'restricted', 'restrictive', 'restricts', 'retaining', 'retarded', 'retired', 'retiring', 'retroactive', 'retrospective', 'reusable',

'reversed', 'reversing', 'revolutionary', 'revolutionised', 'rich', 'ridiculous', 'right', 'risking', 'risky', 'rivalry', 'robust', 'rocks', 'romantic', 'rubbish', 'rude', 'rudimentary', 'rugged', 'ruined',

'ruining', 'rushed',

'sad', 'sadly', 'safe', 'safer', 'same', 'satisfactory', 'satisfied', 'satisfy', 'satisfying', 'save', 'saver', 'saves', 'saving', 'scam', 'scared', 'scares', 'scary', 'sceptical', 'scientific', 'screwed',

'seamlessly', 'secondary', 'secure', 'sedimentary', 'seductive', 'segmented', 'selectable', 'selected', 'selects', 'selfish', 'semi-competitive', 'sensible', 'sensitive', 'serious', 'seriously', 'serviceable',

'severe', 'sexist', 'sexy', 'shaky', 'shallow', 'shame', 'shameful', 'shameless', 'shames', 'sharp', 'shat', 'shines', 'shiny', 'shocking', 'shoddy', 'short', 'shorted', 'shorter', 'shortest', 'shy', 'sick',

'significant', 'significantly', 'silent', 'silly', 'similar', 'simple', 'simpler', 'simplified', 'simplistic', 'sincerely', 'skeptical', 'sketchy', 'slacked', 'sleek', 'slick', 'slim', 'sloppy', 'slow',

'slowed', 'slower', 'slowest', 'sluggish', 'small', 'smaller', 'smart', 'smile', 'smilie', 'smooth', 'smoother', 'smoothly', 'snap', 'snappy', 'sociable', 'soft', 'softened', 'softer', 'solid', 'solidify',

'solitary', 'sophisticated', 'sorry', 'so-so', 'sound', 'spam', 'spammed', 'spamming', 'spammy', 'spams', 'spare', 'special', 'specialised', 'specified', 'spectacular', 'splendid', 'sporadic', 'sportive',

'sporty', 'spot-on', 'stabilize', 'stable', 'standalone', 'standby', 'startles', 'startling', 'steady', 'stealing', 'steals', 'stern', 'sticking', 'sticks', 'sticky', 'stingy', 'stinks', 'straight',

'straightforward', 'straight-forward', 'straining', 'strange', 'strangely', 'strapped', 'streamlined', 'streamlines', 'stress', 'stressed', 'stressful', 'strict', 'strictly', 'strong', 'strongest', 'strongly',

'structured', 'struggle', 'struggled', 'struggles', 'struggling', 'stuck', 'stumbling', 'stunning', 'stupendous', 'stupid', 'stupidest', 'sturdy', 'stylish', 'subscribe', 'succeeds', 'success', 'successful',

'successfully', 'suck', 'sucked', 'sucking', 'sucks', 'sudden', 'suffer', 'sufferfest', 'suffering', 'suffers', 'sufficient', 'suggest', 'suit', 'suitable', 'suited', 'suits', 'sunny',

'super', 'superb', 'superfluous', 'superior', 'supporting', 'surpasses', 'surprise', 'surprised', 'surprising', 'suspicious', 'sweating', 'sweaty', 'sweet', 'symptomatic', 'systematic',

'tailored', 'tame', 'taming', 'technical', 'tedious', 'temperamental', 'temporary', 'tempted', 'tempting', 'terrible', 'terribly', 'terrific', 'teriffic', 'terifying', 'terrifying', 'thank', 'thankful',

'thankfully', 'thanks', 'thank-you', 'thankyou', 'thnx', 'thorough', 'thoroughly', 'thoughtful', 'thrilling', 'thrive', 'thx', 'tidy', 'tied', 'tight', 'tighter', 'tiny', 'tiresome', 'tolerable', 'tolerate',

'tongue-in-cheek', 'tons', 'top', 'top-notch', 'tops', 'torn', 'torture', 'tough', 'tougher', 'traditional', 'transparent', 'treat', 'treats', 'tremendous', 'tremendously', 'trending', 'tricky', 'trouble',

'troublesome', 'true', 'truly', 'trust', 'trusty', 'typical',

'uesfull', 'ugly', 'ultra', 'unable', 'unacceptable', 'unaccounted', 'unavailable', 'unawareness', 'unbeatable', 'unbeknownst', 'unbelievable', 'unchecked', 'unclear', 'uncluttered', 'uncomfortable', 'uncontrolled',

'unconventional', 'unconvinced', 'undead', 'undecided', 'underestimates', 'underfunded', 'underhanded', 'understand', 'understood', 'understandable', 'underweight', 'undisputed', 'unexplainable', 'unfair',

'unfamiliar', 'unfit', 'unfollow', 'unforgivable', 'unforgiving', 'unfortunate', 'unfriendly', 'unhappy', 'unhealthy', 'unhelpful', 'unimpressed', 'uninspired', 'uninstall', 'uninstalled', 'uninstalling',

'unintelligible', 'uninteresting', 'unintrusive', 'unintuitive', 'unique', 'unknown', 'unleash', 'unlimited', 'unmatched', 'unmotivated', 'unnatural', 'unnecessary', 'unlikely', 'unnecessarily', 'unneeded',

'unobtrusive', 'unofficial', 'unorganized', 'unpaid', 'unpause', 'unprepared', 'unprofessional', 'unreadable', 'un-relevant', 'unrealistic', 'unrelated', 'unreliable', 'unrivaled', 'unsafe', 'unsatisfactory',

'unspecified', 'unstable', 'unsurpassed', 'untested', 'unusable', 'unusual', 'unwanted', 'unwieldy', 'uplifting', 'upper', 'upset', 'upsets', 'urban', 'urgent', 'usable', 'useable', 'useful', 'usefull',

'usefully', 'useless', 'user-friendly', 'usless',
'vague', 'valid', 'valuable', 'values', 'vast', 'versatile', 'viable', 'vibrant', 'viewable', 'viligent', 'virtual', 'visible', 'vital', 'vitals', 'vivid', 'vividly', 'vulnerable',

'wan', 'want', 'wants', 'waste', 'wasted', 'wasteful', 'wastes', 'watchful', 'wearable', 'weird', 'well', 'welcoming', 'well-done', 'well-designed', 'well-organized', 'well-thought', 'well-written', 'wholistic',

'wicked', 'wide', 'wider', 'wierd', 'wild', 'winner', 'wise', 'wish', 'wished', 'wobbly', 'won', 'wonder', 'wonderful', 'wonderfull', 'wonderfully', 'wondering', 'wonky', 'work', 'workable', 'working', 'works',

'worldwide', 'worried', 'worries', 'worse', 'worst', 'worth', 'worthfull', 'worthless', 'worthwhile', 'worthy', 'wounded', 'wow', 'wrong', 'wrongly'

13 Appendix F OOPS!'s evaluation result for GMAURO

Evaluation results:

It is obvious that not all the pitfalls are equally important; their impact in the ontology will depend on multiple factors. For this reason, each pitfall has an importance level attached indicating how important it is. We have identified three levels:

- **Critical** 🚫 : It is crucial to correct the pitfall. Otherwise, it could affect the ontology consistency, reasoning, applicability, etc.
- **Important** ⚠️ : Though not critical for ontology function, it is important to correct this type of pitfall.
- **Minor** 🟡 : It is not really a problem, but by correcting it we will make the ontology nicer.

[\[Expand All\]](#) | [\[Collapse All\]](#)

Results for P08: Missing annotations.89 cases | Minor 🟡

This pitfall consists in creating an ontology element and failing to provide human readable annotations attached to it. Consequently, ontology elements lack annotation properties that label them (e.g. rdfs:label, lemon:LexicalEntry, skos:prefLabel or skos:altLabel) or that define them (e.g. rdfs:comment or dc:description). This pitfall is related to the guidelines provided in [5].

- The following elements have neither rdfs:label or rdfs:comment defined:
 - > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#MobileDevice>
 - > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#App>
 - > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#DatePickerDialog>
 - > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#Feature>
 - > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#Font>
 - > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#Sensors>
 - > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#TouchInput>
 - > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#Category>
 - > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#SourceImage>
 - > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#Request>
 - > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#Request>

[rsion14#Setting](#)

> [rsion14#Button](http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVe)

[rsion14#Button](#)

> [rsion14#PopupMenu](http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVe)

[rsion14#PopupMenu](#)

> [rsion14#Device](http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVe)

[rsion14#Device](#)

> [rsion14#Price](http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVe)

[rsion14#Price](#)

> [rsion14#UserLocation](http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVe)

[rsion14#UserLocation](#)

> [rsion14#NegativeReview](http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVe)

[rsion14#NegativeReview](#)

> [rsion14#Subject](http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVe)

[rsion14#Subject](#)

> [rsion14#Function](http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVe)

[rsion14#Function](#)

> [rsion14#PopupMenu](http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVe)

[rsion14#PopupMenu](#)

> [rsion14#Toast](http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVe)

[rsion14#Toast](#)

> [rsion14#ImageGraphics](http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVe)

[rsion14#ImageGraphics](#)

> [rsion14#Version](http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVe)

[rsion14#Version](#)

> [rsion14#Opinion](http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVe)

[rsion14#Opinion](#)

> [rsion14#Search](http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVe)

[rsion14#Search](#)

> [rsion14#Color](http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVe)

[rsion14#Color](#)

> [rsion14#Menu](http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVe)

[rsion14#Menu](#)

> [rsion14#Attribute](http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVe)

[rsion14#Attribute](#)

> [rsion14#NeutralReview](http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVe)

[rsion14#NeutralReview](#)

> [rsion14#TimePickerDialog](http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVe)

[rsion14#TimePickerDialog](#)

> [rsion14#UserInterface](http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVe)

[rsion14#UserInterface](#)

- > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#Background>
- > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#Notification>
- > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#Camera>
- > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#AlertDialog>
- > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#GlucoseMonitor>
- > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#PositiveReview>
- > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#Connectivity>
- > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#Scrollbar>
- > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#Screen>
- > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#AudioVideo>
- > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#Review>
- > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#Dialog>
- > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#OptionsMenu>
- > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#Animations>
- > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#Size>
- > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#Text>
- > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#ContextualMenu>
- > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasScrollbar>
- > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#reviewHasOpinion>
- > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasSourceImage>
- > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasSourceImage>

[rsion14#subjectIsRegardedAsOpinion](#)

> <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#opinionIsRegardingSubject>

> <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasFont>

> <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#isReviewOfApp>

> <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#appHasReview>

> <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasText>

> <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasBackground>

> <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#isOpinionFromReview>

> <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#subjectAppearedInReview>

> <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasColor>

> <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasButton>

> <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#categoryHasAttribute>

> <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#reviewIsTalkingAboutSubject>

> <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasSize>

> <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasAppUpdatedOnDate>

> <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasSentimentScore>

> <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasAppInAppProductsPricing>

> <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasAppLink>

> <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasAppDeveloperEmail>

> <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasAppNumberOfRatings>

> <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasAppContentRating>

- > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasAppCurrentVersion>
- > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasAppDeveloperWebsite>
- > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasAppName>
- > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasReviewDate>
- > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasAppRequiresAndroid>
- > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasRequestText>
- > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasReviewRating>
- > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasAppSize>
- > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasAppDeveloperCompany>
- > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasAuthorName>
- > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasTitle>
- > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasAppNumberOfInstalls>
- > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasAppPricing>
- > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#keywords>
- > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasNumberOfHelpful>
- > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasReviewText>
- > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasAppOverallRating>

Results for P11: Missing domain or range in properties.32 cases | Important 🟡

Object and/or datatype properties without domain or range (or none of them) are included in the ontology.

- This pitfall appears in the following elements:
- > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasAppCurrentVersion>

[rsion14#hasSize](#)

> <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#categoryHasAttribute>

> <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasButton>

> <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasColor>

> <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasBackground>

> <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasText>

> <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasFont>

> <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasSourceImage>

> <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasScrollbar>

> <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasAppOverallRating>

> <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasReviewText>

> <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasNumberOfHelpful>

> <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#keywords>

> <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasAppPricing>

> <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasAppNumberOfInstalls>

> <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasTitle>

> <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasAuthorName>

> <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasAppDeveloperCompany>

> <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasAppSize>

> <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasReviewRating>

> <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasRequestText>

- > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasAppRequiresAndroid>
- > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasReviewDate>
- > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasAppName>
- > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasAppDeveloperWebsite>
- > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasAppCurrentVersion>
- > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasAppContentRating>
- > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasAppNumberOfRatings>
- > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasAppDeveloperEmail>
- > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasAppLink>
- > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasAppInAppProductsPricing>
- > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasAppUpdatedOnDate>

• **Tip:** Solving this pitfall may lead to new results for other pitfalls and suggestions. We encourage you to solve all cases when needed and see what else you can get from OOPS!

Results for P13: Inverse relationships not explicitly declared.9 cases | Minor 🟡

This pitfall appears when any relationship (except for those that are defined as symmetric properties using owl:SymmetricProperty) does not have an inverse relationship (owl:inverseOf) defined within the ontology.

• This pitfall appears in the following elements:

- > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasScrollbar>
- > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasSourceImage>
- > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasFont>
- > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasText>
- > <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasAppUpdatedOnDate>

[rsion14#hasBackground](#)

> <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasColor>

> <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasButton>

> <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#categoryHasAttribute>

> <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#hasSize>

Results for P30: Equivalent classes not explicitly declared.1 case | Important 🟡

This pitfall consists in missing the definition of equivalent classes (owl:equivalentClass) in case of duplicated concepts. When an ontology reuses terms from other ontologies, classes that have the same meaning should be defined as equivalent in order to benefit the interoperability between both ontologies.

- The following classes might be equivalent:

> <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#Background>, <http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#Setting>

Results for P41: No license declared.ontology* | Important 🟡

The ontology metadata omits information about the license that applies to the ontology.

- This pitfall appears in the following elements:

> [It has no licence defined](#)

14 Appendix G Answers to the SPARQL queries about the applications of the GMAURO ontology

Q1. Regarding one aspect, what topics have people mentioned?

An example SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX generalMobileAppUserReviewOntologyVersion14:
<http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#>
SELECT ?category WHERE {
  ?category rdf:type generalMobileAppUserReviewOntologyVersion14:UserInterface.
}
ORDER BY ASC(?category)
```

A screenshot of an example result returned by Snap SPARQL Query from dataset one:

Evaluated BGP in 6 ms

generalMobileAppUserReviewOntologyVersion14:activities
generalMobileAppUserReviewOntologyVersion14:activity
generalMobileAppUserReviewOntologyVersion14:background
generalMobileAppUserReviewOntologyVersion14:backgrounds
generalMobileAppUserReviewOntologyVersion14:button
generalMobileAppUserReviewOntologyVersion14:buttons
generalMobileAppUserReviewOntologyVersion14:color
generalMobileAppUserReviewOntologyVersion14:colors
generalMobileAppUserReviewOntologyVersion14:crashes
generalMobileAppUserReviewOntologyVersion14:display
generalMobileAppUserReviewOntologyVersion14:displays
generalMobileAppUserReviewOntologyVersion14:fields
generalMobileAppUserReviewOntologyVersion14:font
generalMobileAppUserReviewOntologyVersion14:fonts
generalMobileAppUserReviewOntologyVersion14:giggling
generalMobileAppUserReviewOntologyVersion14:green
generalMobileAppUserReviewOntologyVersion14:grey
generalMobileAppUserReviewOntologyVersion14:gui
generalMobileAppUserReviewOntologyVersion14:icon
generalMobileAppUserReviewOntologyVersion14:icons
generalMobileAppUserReviewOntologyVersion14:interface
generalMobileAppUserReviewOntologyVersion14:interfaces

Figure 34 A screenshot of an example result of SPARQL query Q1

Q2. What are the opinions of users on topics in this category?

An example SPARQL query:

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX generalMobileAppUserReviewOntologyVersion14:

<http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#>

```

SELECT ?category ?opinions WHERE {
  ?category rdf:type generalMobileAppUserReviewOntologyVersion14:UserInterface.
  ?opinions rdf:type generalMobileAppUserReviewOntologyVersion14:Opinion.
  ?opinions generalMobileAppUserReviewOntologyVersion14:opinionIsRegardingSubject
?category
}
ORDER BY ASC(?category)

```

A screenshot of an example result returned by Snap SPARQL Query from dataset one:

Evaluated BGP in 876 ms

?category	?opinions
generalMobileAppUserReviewOntologyVersion14:activities	generalMobileAppUserReviewOntologyVersion14:different
generalMobileAppUserReviewOntologyVersion14:activities	generalMobileAppUserReviewOntologyVersion14:love
generalMobileAppUserReviewOntologyVersion14:activities	generalMobileAppUserReviewOntologyVersion14:active
generalMobileAppUserReviewOntologyVersion14:activity	generalMobileAppUserReviewOntologyVersion14:busy
generalMobileAppUserReviewOntologyVersion14:background	generalMobileAppUserReviewOntologyVersion14:lighter
generalMobileAppUserReviewOntologyVersion14:background	generalMobileAppUserReviewOntologyVersion14:outdated
generalMobileAppUserReviewOntologyVersion14:background	generalMobileAppUserReviewOntologyVersion14:prefer
generalMobileAppUserReviewOntologyVersion14:background	generalMobileAppUserReviewOntologyVersion14:wish
generalMobileAppUserReviewOntologyVersion14:backgrounds	generalMobileAppUserReviewOntologyVersion14:better
generalMobileAppUserReviewOntologyVersion14:button	generalMobileAppUserReviewOntologyVersion14:save
generalMobileAppUserReviewOntologyVersion14:button	generalMobileAppUserReviewOntologyVersion14:simple
generalMobileAppUserReviewOntologyVersion14:button	generalMobileAppUserReviewOntologyVersion14:big
generalMobileAppUserReviewOntologyVersion14:button	generalMobileAppUserReviewOntologyVersion14:uninstall
generalMobileAppUserReviewOntologyVersion14:button	generalMobileAppUserReviewOntologyVersion14:hitting
generalMobileAppUserReviewOntologyVersion14:buttons	generalMobileAppUserReviewOntologyVersion14:big
generalMobileAppUserReviewOntologyVersion14:buttons	generalMobileAppUserReviewOntologyVersion14:intuitive
generalMobileAppUserReviewOntologyVersion14:color	generalMobileAppUserReviewOntologyVersion14:lighter
generalMobileAppUserReviewOntologyVersion14:color	generalMobileAppUserReviewOntologyVersion14:new

Figure 35 A screenshot of an example result of SPARQL query Q2

Q3. What are the opinions of people on a single topic?

An example SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
```



```

PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX generalMobileAppUserReviewOntologyVersion14:
<http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserRevi
ewOntologyVersion14#>
SELECT ?topic ?opinions WHERE {
  ?topic rdfs:type generalMobileAppUserReviewOntologyVersion14:UserInterface.
  ?opinions rdfs:type generalMobileAppUserReviewOntologyVersion14:Opinion.
  ?opinions generalMobileAppUserReviewOntologyVersion14:opinionIsRegardingSubject
?topic.
  filter(?topic =
<http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserRevi
ewOntologyVersion14#font>)
}
ORDER BY ASC(?category)

```

A screenshot of an example result returned by Snap SPARQL Query from dataset one:

Evaluated BGP in 545 ms

?topic	?opinions
generalMobileAppUserReviewOntologyVersion14:font	generalMobileAppUserReviewOntologyVersion14:tiny
generalMobileAppUserReviewOntologyVersion14:font	generalMobileAppUserReviewOntologyVersion14:big
generalMobileAppUserReviewOntologyVersion14:font	generalMobileAppUserReviewOntologyVersion14:small
generalMobileAppUserReviewOntologyVersion14:font	generalMobileAppUserReviewOntologyVersion14:light
generalMobileAppUserReviewOntologyVersion14:font	generalMobileAppUserReviewOntologyVersion14:hard

Figure 36 A screenshot of an example result of SPARQL query Q3

Q4. How many reviews are there in each mobile app?

An example SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX generalMobileAppUserReviewOntologyVersion14:
<http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#>
SELECT ?app (count(distinct ?reviews) as ?count) WHERE {
  ?app rdf:type generalMobileAppUserReviewOntologyVersion14:App.
  ?reviews rdf:type generalMobileAppUserReviewOntologyVersion14:Review.
  ?reviews generalMobileAppUserReviewOntologyVersion14:isReviewOfApp ?app.
}
group by ?app
order by ASC(?app)
```

A screenshot of an example result returned by Snap SPARQL Query from dataset one:

Evaluated BGP in 8360 ms

?app	?count
generalMobileAppUserReviewOntologyVersion14:app1	181
generalMobileAppUserReviewOntologyVersion14:app10	467
generalMobileAppUserReviewOntologyVersion14:app11	222
generalMobileAppUserReviewOntologyVersion14:app12	506
generalMobileAppUserReviewOntologyVersion14:app13	68
generalMobileAppUserReviewOntologyVersion14:app14	249
generalMobileAppUserReviewOntologyVersion14:app15	21
generalMobileAppUserReviewOntologyVersion14:app16	550
generalMobileAppUserReviewOntologyVersion14:app17	1532
generalMobileAppUserReviewOntologyVersion14:app18	243
generalMobileAppUserReviewOntologyVersion14:app2	232
generalMobileAppUserReviewOntologyVersion14:app3	6081
generalMobileAppUserReviewOntologyVersion14:app4	3300
generalMobileAppUserReviewOntologyVersion14:app5	6323
generalMobileAppUserReviewOntologyVersion14:app6	361
generalMobileAppUserReviewOntologyVersion14:app7	300
generalMobileAppUserReviewOntologyVersion14:app8	205
generalMobileAppUserReviewOntologyVersion14:app9	439

Figure 37 A screenshot of an example result of SPARQL query Q4

Q5. What topics did users mention in a mobile app?

Unacceptably slow on a normal computer, therefore it has to be forced to quit.

Q6. Which topics do people have most opinions about in a certain category, such as UserInterface, Feature, Function, etc.?

An example SPARQL query:

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX generalMobileAppUserReviewOntologyVersion14:

<http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#>

```
SELECT ?category (count (distinct ?opinions) as ?count) WHERE {  
  ?category rdf:type generalMobileAppUserReviewOntologyVersion14:UserInterface.  
  ?opinions rdf:type generalMobileAppUserReviewOntologyVersion14:Opinion.  
  ?opinions generalMobileAppUserReviewOntologyVersion14:opinionIsRegardingSubject  
  ?category.  
}  
group by ?category  
order by DESC(?count )
```

A screenshot of an example result returned by Snap SPARQL Query from dataset one:

Evaluated BGP in 581 ms

?category	?count
generalMobileAppUserReviewOntologyVersion14:interface	61
generalMobileAppUserReviewOntologyVersion14:ui	27
generalMobileAppUserReviewOntologyVersion14:screen	21
generalMobileAppUserReviewOntologyVersion14:layout	19
generalMobileAppUserReviewOntologyVersion14:view	15
generalMobileAppUserReviewOntologyVersion14:stuff	15
generalMobileAppUserReviewOntologyVersion14:display	9
generalMobileAppUserReviewOntologyVersion14:sounds	9
generalMobileAppUserReviewOntologyVersion14:scroll	8
generalMobileAppUserReviewOntologyVersion14:gui	8
generalMobileAppUserReviewOntologyVersion14:sound	7
generalMobileAppUserReviewOntologyVersion14:color	7
generalMobileAppUserReviewOntologyVersion14:messages	7
generalMobileAppUserReviewOntologyVersion14:screens	7
generalMobileAppUserReviewOntologyVersion14:settings	6
generalMobileAppUserReviewOntologyVersion14:search	6
generalMobileAppUserReviewOntologyVersion14:presentation	6
generalMobileAppUserReviewOntologyVersion14:fields	6

Figure 38 A screenshot of an example result of SPARQL query Q6

Q7. For all topics, how many opinions are on each topic?

An example SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
```

```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
```

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
```

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
```

```
PREFIX generalMobileAppUserReviewOntologyVersion14:
```

```
<http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#>
```

```
SELECT ?topic (count ( ?opinions) as ?count) WHERE {
```

```
?topic rdf:type ?topics.
```

```
?topics rdfs:subClassOf generalMobileAppUserReviewOntologyVersion14:Subject.
```

```
?opinions rdf:type generalMobileAppUserReviewOntologyVersion14:Opinion.
```

```
?opinions generalMobileAppUserReviewOntologyVersion14:opinionIsRegardingSubject
```

```
?topic.
```

```
}
```

```
group by ?topic
```

```
order by DESC(?count )
```

A screenshot of an example result returned by Snap SPARQL Query from dataset one:

Evaluated BGP in 74805 ms

?topic	?count
generalMobileAppUserReviewOntologyVersion14:app	1035
generalMobileAppUserReviewOntologyVersion14:it	385
generalMobileAppUserReviewOntologyVersion14:work	207
generalMobileAppUserReviewOntologyVersion14:track	172
generalMobileAppUserReviewOntologyVersion14:use	154
generalMobileAppUserReviewOntologyVersion14:record	102
generalMobileAppUserReviewOntologyVersion14:way	94
generalMobileAppUserReviewOntologyVersion14:log	85
generalMobileAppUserReviewOntologyVersion14:version	69
generalMobileAppUserReviewOntologyVersion14:application	65
generalMobileAppUserReviewOntologyVersion14:tracking	61
generalMobileAppUserReviewOntologyVersion14:information	61
generalMobileAppUserReviewOntologyVersion14:reports	60
generalMobileAppUserReviewOntologyVersion14:see	60
generalMobileAppUserReviewOntologyVersion14:program	60
generalMobileAppUserReviewOntologyVersion14:records	58
280 results	

Figure 39 A screenshot of an example result of SPARQL query Q7

Q8. For all topics, how many people mentioned about each topic?

An example SPARQL query:

```

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX generalMobileAppUserReviewOntologyVersion14:
<http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#>
SELECT ?topic (count ( ?reviews) as ?count) WHERE {
?topic rdf:type ?topics.
?topics rdfs:subClassOf generalMobileAppUserReviewOntologyVersion14:Subject.
?reviews rdf:type generalMobileAppUserReviewOntologyVersion14:Review.
?reviews generalMobileAppUserReviewOntologyVersion14:reviewIsTalkingAboutSubject
?topic.

```

}

group by ?topic

order by DESC(?count)

A screenshot of an example result returned by Snap SPARQL Query from dataset one:

Evaluated BGP in 960353 ms

?topic	?count
generalMobileAppUserReviewOntologyVersion14:app	12357
generalMobileAppUserReviewOntologyVersion14:it	3283
generalMobileAppUserReviewOntologyVersion14:track	3236
generalMobileAppUserReviewOntologyVersion14:use	2759
generalMobileAppUserReviewOntologyVersion14:work	774
generalMobileAppUserReviewOntologyVersion14:way	487
generalMobileAppUserReviewOntologyVersion14:tool	370
generalMobileAppUserReviewOntologyVersion14:record	354
generalMobileAppUserReviewOntologyVersion14:works	264
generalMobileAppUserReviewOntologyVersion14:version	251
generalMobileAppUserReviewOntologyVersion14:see	247
generalMobileAppUserReviewOntologyVersion14:log	239
generalMobileAppUserReviewOntologyVersion14:tracking	238
generalMobileAppUserReviewOntologyVersion14:application	205
generalMobileAppUserReviewOntologyVersion14:features	199
generalMobileAppUserReviewOntologyVersion14:help	161
2948 results	

Figure 40 A screenshot of an example result of SPARQL query Q8

(The Reason for this answer being bigger than the Q7 is that it contains 2668 user requests.)

Q9. How many people have expressed opinions on a single topic?

An example SPARQL query:

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

265

```

PREFIX generalMobileAppUserReviewOntologyVersion14:
<http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#>

SELECT ?topic (count (distinct ?reviews) as ?count) WHERE {
  ?topic rdf:type generalMobileAppUserReviewOntologyVersion14:Version.
  ?reviews rdf:type generalMobileAppUserReviewOntologyVersion14:Review.
  ?reviews generalMobileAppUserReviewOntologyVersion14:reviewIsTalkingAboutSubject
?topic.
  filter (?topic =
<http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#version>)
}
group by ?topic

```

A screenshot of an example result returned by Snap SPARQL Query from dataset one:

Evaluated BGP in 406 ms

?topic	?count
generalMobileAppUserReviewOntologyVersion14:version	251

Figure 41 A screenshot of an example result of SPARQL query Q9

Q10. What are the opinions on a single topic category, and how many people think about each opinion?

An example SPARQL query:

```

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

```


PREFIX generalMobileAppUserReviewOntologyVersion14:

<http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#>

```
SELECT ?topics ?opinions (count (distinct ?reviews) as ?count) WHERE {  
  ?topics rdf:type generalMobileAppUserReviewOntologyVersion14:Version.  
  ?reviews rdf:type generalMobileAppUserReviewOntologyVersion14:Review.  
  ?opinions rdf:type generalMobileAppUserReviewOntologyVersion14:Opinion.  
  ?opinions generalMobileAppUserReviewOntologyVersion14:opinionIsRegardingSubject  
  ?topics.  
  ?reviews generalMobileAppUserReviewOntologyVersion14:reviewHasOpinion ?opinions.  
}  
group by ?topics ?opinions  
order by ASC (?topics) DESC(?count )
```

A screenshot of an example result returned by Snap SPARQL Query from dataset one:

Evaluated BGP in 997850 ms

?topics	?opinions	?count
generalMobileAppUserReviewOntologyVersion14:news	generalMobileAppUserReviewOntologyVersion14:great	3124
generalMobileAppUserReviewOntologyVersion14:news	generalMobileAppUserReviewOntologyVersion14:latest	63
generalMobileAppUserReviewOntologyVersion14:news	generalMobileAppUserReviewOntologyVersion14:bad	38
generalMobileAppUserReviewOntologyVersion14:update	generalMobileAppUserReviewOntologyVersion14:easy	3334
generalMobileAppUserReviewOntologyVersion14:update	generalMobileAppUserReviewOntologyVersion14:great	3124
generalMobileAppUserReviewOntologyVersion14:update	generalMobileAppUserReviewOntologyVersion14:love	2007
generalMobileAppUserReviewOntologyVersion14:update	generalMobileAppUserReviewOntologyVersion14:good	1413
generalMobileAppUserReviewOntologyVersion14:update	generalMobileAppUserReviewOntologyVersion14:like	1121
generalMobileAppUserReviewOntologyVersion14:update	generalMobileAppUserReviewOntologyVersion14:helps	718
generalMobileAppUserReviewOntologyVersion14:update	generalMobileAppUserReviewOntologyVersion14:better	512
generalMobileAppUserReviewOntologyVersion14:update	generalMobileAppUserReviewOntologyVersion14:helpful	470
generalMobileAppUserReviewOntologyVersion14:update	generalMobileAppUserReviewOntologyVersion14:awesome	447
generalMobileAppUserReviewOntologyVersion14:update	generalMobileAppUserReviewOntologyVersion14:works	425
generalMobileAppUserReviewOntologyVersion14:update	generalMobileAppUserReviewOntologyVersion14:keeps	338
generalMobileAppUserReviewOntologyVersion14:update	generalMobileAppUserReviewOntologyVersion14:new	275
generalMobileAppUserReviewOntologyVersion14:update	generalMobileAppUserReviewOntologyVersion14:able	225
generalMobileAppUserReviewOntologyVersion14:update	generalMobileAppUserReviewOntologyVersion14:perfect	193
generalMobileAppUserReviewOntologyVersion14:update	generalMobileAppUserReviewOntologyVersion14:want	164
146 results		

Figure 42 A screenshot of an example result of SPARQL query Q10

(However, the last column “?count” is believed to be wrong. This is explained as in section 4.6 of the Results and evaluation chapter.)

Q11. In the reviews that mention topics in a certain category, how many people have left positive reviews for each topic (or how many people have left negative reviews, and how many people are neutral)?

An example SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX generalMobileAppUserReviewOntologyVersion14:
<http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#>
SELECT ?topics (count (distinct ?reviews) as ?count) WHERE {
  ?topics rdf:type generalMobileAppUserReviewOntologyVersion14:Version.
  ?reviews rdf:type generalMobileAppUserReviewOntologyVersion14:Review.
  ?reviews generalMobileAppUserReviewOntologyVersion14:reviewsIsTalkingAboutSubject
?topics.
  ?reviews generalMobileAppUserReviewOntologyVersion14:hasSentimentScore ?score.
  filter(?score > 0).
}
group by ?topics
order by DESC(?count )
```

A screenshot of an example result returned by Snap SPARQL Query from dataset one:

Evaluated BGP in 620 ms
268

?topics	?count
generalMobileAppUserReviewOntologyVersion14:version	184
generalMobileAppUserReviewOntologyVersion14:update	70
generalMobileAppUserReviewOntologyVersion14:updates	11
generalMobileAppUserReviewOntologyVersion14:news	2

Figure 43 A screenshot of an example result of SPARQL query Q11

Q12. In a mobile phone application, how many people have left positive reviews (or how many people have left negative reviews, and how many people are neutral)?

An example SPARQL query:

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX generalMobileAppUserReviewOntologyVersion14:

<http://www.semanticweb.org/juanwang/ontologies/2018/7/generalMobileAppUserReviewOntologyVersion14#>

SELECT (count (distinct ?reviews) as **?positiveCount**) WHERE {

?reviews rdf:type generalMobileAppUserReviewOntologyVersion14:Review.

?reviews generalMobileAppUserReviewOntologyVersion14:hasSentimentScore ?score.

filter(?score > 0).

}

A screenshot of an example result returned by Snap SPARQL Query from dataset one:

Evaluated BGP in 317 ms

?positiveCount
14060

Figure 44 A screenshot of an example result of SPARQL query Q12